Automatic Image Tagging

Sistemas Informaticos
supervised by

High Performance Computing Lab.
National Research Council (CNR), Italia
and
Facultad de Informatica,
Universidad Complutense (UCM)

Claudio Lucchese,
Raffaele Perego and
Jose Ignacio Hidalgo

Author:	Ignacio Arnaldo Lucas
E-Mail:	nachoarnaldo@gmail.com
Closing date:	July 1, 2010

Automatic Image Tagging

Ignacio Arnaldo Lucas

July 3, 2010

1 Abstract

Las aplicaciones web como ImageShack, Flickr o Facebook que permiten a los usuarios compartir sus imágenes son cada vez mas populares. Cuando el usuario comparte una imagen, se le pide rellenar diferentes campos textuales como título, tags y comentarios entre otros. Esta tarea es aburrida y frustrante a largo plazo y como resultado, la información aportada es escasa y de mala calidad. Esto es perjudicial para los sistemas de búsqueda que explotan la información aportada por el usuario para realizar búsquedas textuales. Un sistema recomendador de tags mejoraría notablemente la calidad de los tags y, por lo tanto, se podrían implementar mejores sistemas de búsqueda en estas páginas. Hemos desarrollado un recomendador de tags que, dada una imagen, propone diez tags. El sistema recomendador ha sido entrenado con un gran conjunto de imágenes obtenidas de Flickr. Proponemos nuevos algoritmos que combinan la utilización de características tanto globales como locales de la imagen (descriptores MPEG-7 y SURF), asi como técnicas de clustering para afrontar el problema de la recomendación de tags en un tiempo aceptable.

Palabras clave

imagen, tag, recomendador, contenido, búsqueda, reconocimiento, objeto, clustering

Automatic Image Tagging

Ignacio Arnaldo Lucas

July 3, 2010

1 Abstract

Web applications such as ImageShack, Flickr or Facebook that allow users to share their images have become extremely popular. When an image is uploaded, the user is asked to add title, tags, comments and other information. This task is annoying and frustrating for the user and, as a result, little and low quality data is provided. This is harmful for retrieval systems which exploit user annotation to support textual searchs. A tag recommender would certainly improve the quality of the tags, hence better retrieval systems could be implemented in these pages. We implemented a tag recommender that proposes ten tags for a given query image. The recommender is trained with a large dataset of over 100000 annotated images crawled from Flickr. We propose new algorithms that combine the use of both global and local features (MPEG-7 and SURF descriptors) and clustering techniques in order to address the tag recommendation problem in an acceptable time.

Keywords
image, tag, recommender, content-based, retrieval, object, recognition, clustering

Automatic Image Tagging

Ignacio Arnaldo Lucas

July 1, 2010

Contents

1 Introduction **3**
- 1.1 Motivations . . . 3
- 1.2 Related work . . . 4
- 1.3 Challenges and techniques . . . 5
- 1.4 Contribution of this thesis . . . 6

2 Similarity matching based on MPEG-7 features **8**
- 2.1 Scalable color type descriptor . . . 10
- 2.2 Color layout type descriptor . . . 11
- 2.3 Color structure type descriptor . . . 12
- 2.4 Dominant color type descriptor . . . 13
- 2.5 Homogeneous texture type descriptor . . . 14
- 2.6 Edge histogram type descriptor . . . 15

3 Similarity matching based on SURF keypoints **17**
- 3.1 Scale Invariant Feature Transform . . . 17
  - 3.1.1 SIFT features extraction . . . 17
  - 3.1.2 Matching of the SIFT feature vectors . . . 19
- 3.2 SIFT vs SURF . . . 21
- 3.3 Distances and metrics used with SURF features . . . 24
  - 3.3.1 Search of the nearest match . . . 25
  - 3.3.2 Search of the n nearest matches . . . 25
  - 3.3.3 Amount of matches . . . 25
  - 3.3.4 Amount of bimatches . . . 25

4 Algorithms proposed **26**
- 4.1 Cleaning of the dataset . . . 26
- 4.2 Metrics . . . 28
  - 4.2.1 Tagging accuracy . . . 28
  - 4.2.2 Classification accuracy . . . 28
- 4.3 First approach . . . 29
- 4.4 Baseline . . . 32
- 4.5 Textual + Visual Clustering . . . 36
  - 4.5.1 Textual Clustering . . . 36
  - 4.5.2 Visual Clustering . . . 41
  - 4.5.3 Tuning the number of visual clusters . . . 43
  - 4.5.4 Freq. cluster visuale vs freq. cluster text . . . 45
  - 4.5.5 Visual Cluster Filter . . . 46
  - 4.5.6 Examples . . . 50

5 Visual Clusters + SURF Features **53**
5.1 COST ANALYSIS 53
5.2 Candidate Selection

54
5.2.1 Candidate Selection Strategy 1 55
5.2.2 Candidate Selection Strategy 2: Looking in the Visual Clusters 57
5.3 Study of the relation between SURF Bimatches and Exact Matching 60
5.4 Comparing D_1, D_3 and D_5 64
5.5 Finding the Optimal Threshold for SURF Bimatches 64

6 Improvements **67**
6.1 Tag cooccurrence 67
6.2 Implementation Improvement 68
6.2.1 Metric Spaces 68
6.2.2 Use of the M-tree structure 69

7 Big Scale Application **70**
7.1 Cleaning of the dataset 70
7.2 Textual Clustering 70
7.3 Visual Clustering 71
7.4 SURF keypoints 71
7.5 Strategy followed 75
7.6 Examples 76

8 Conclusions **78**

Chapter 1

Introduction

1.1 Motivations

Web 2.0 sites such as YouTube, ImageShack, last.fm, Flickr or Facebook that allow users to share their content have become extremely popular. There has been a rise in traffic and contribution levels on most of these sites in the last years. For example, Facebook claims to have more than 400 million active users and that 50% of these users log into the site each day. In this social networking site, more than 700 million photos are uploaded monthly. Another example is Flickr, the site that allows users to upload, comment, organize and share digital photos. In 2008, Flickr passed from 2 to 3 billion hosted images, showing a rapid increase in photo sharing. This site now claims to host more than 4 billion images.

In Web 2.0 sites, the multimedia content usually comes with a title, tags, comments and other information that is mostly written by the user. We think that writing all this data is annoying and frustrating and, as a result, little and low quality data is provided. This is harmful for retrieval systems which exploit user annotation to support textual searchs. In fact, many Flickr images are bad tagged or don't have any tag at all, this means that these hosted images are not searchable. In the figure 1.1, whe show the all time Flickr most popular tags. Among these tags we can find 'geotagged', 'nikon' or 'photography' which are obviously not related to the visual content of the images.

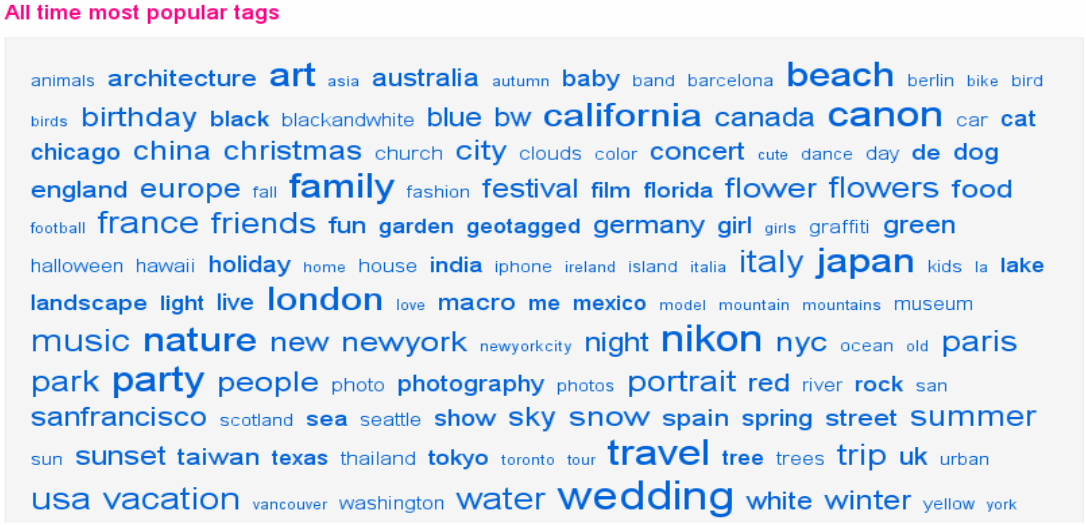


Figure 1.1: All time Flickr most popular tags

We need a recommender system for helping the user in tagging photos. In fact, a tag recommender would certainly improve the quality of the tags, hence better retrieval systems could be implemented in these pages. Our pourpose with this work is to recommend tags to the users when they are uploading their images. Our recommender could also be used in a hidden way in order to enrich the text associated to some multimedia content.

1.2 Related work

Recommender systems are applications that present information items to the users in which they might be interested. These systems usually work with large scale applications. The items proposed may be web pages, images, videos, news etc ... There are mainly two approaches: the collaborative filtering approach and the content-based approach. The collaborative filtering recommendation engines are based on the user's profile and his social environment. Many algorithms and techniques have been proposed and are currently used in web pages such as Amazon, eBay, Last.fm and many more. The content-based approach is a larger field of research because this approach depends on the kind of data we are working with. Most of the works proposed in this field exploit information from textual data.

In particular, the tag recommendation problem has already been studied but usually, a collaborative filtering approach is used. In the paper 'Flickr Tag Recommendation based on Collective Knowledge' a strategy based on tag cooccurrence is used in order to propose tags. Other articles find in k-partite graphs a model to the recommendation problem. In particular, different partitioning and clustering techniques are studied for folksonomies, wich are used to modelize the relation between content, tags and users. These folksonomies are the result of personal free tagging of information and objects where the tagging is done in a social environment (shared and open to others). When people tag items, they are providing their meaning in their own understanding and offering a means to connect these items.
In this work, we follow a content-based approach in order to recommend tags. In this case, we are interested in the study of visual features of the images, therefore our work has many common points with the techniques used in content-based image retrieval. Many experiments done in this field use MPEG-7 descriptors, for example in 'Building a Web-scale Image Similarity Search System' a search engine is built based on MPEG-7 features. A combinantion of MPEG-7 descriptors of the query image is used to retrieve similar images. As we explain in this section, these features have been widely used, but usually with the goal of classifying an image in a reduced set of categories. The paper 'Adaptive Committees of Feature-specific Classifiers for Image Classification' uses several combinations of these descriptors in order to classify stone slabs in 37 different categories. Other features are used in CBIR, we are specially interested in local features like SIFT descriptors. Introduced by David G. Lowe, these features are used in object recognition and should help identify concrete places and objects. In the article 'How Flickr Helps us Make Sense of the World: Context and Content in Community-Contributed Media Collections' a clustering technique based on geo-references, colors, textures and SIFT features is used to retrieve representative images of San Francisco. The same idea is used in the papers 'Placing Flickr Photos on a Map' and 'Mapping the World's Photos' where visual features help locating a set of flickr images geographically. In these cases, classifiers are trained with geotagged images and its visual and textual features (SIFT and tags). The paper 'Combining Image Descriptors to Effectively Retrieve Events from Visual Lifelogs' uses MPEG-7, SIFT and SURF descriptors to classify images taken from a wearable camera into many different events of normal life. They work with a big dataset so this is the most similar work that we have found.
A lot of work has been done concerning image retrieval, nevertheless extracting semantic meaning from images is an open area in research. Techniques like image segmentation have been proposed, for example in the paper 'Automatic Image Annotation and Retrieval using CrossMedia Relevance Models' segments are extracted from the images. These segments are then regrouped into clusters called 'blobs' which should have a common semantic meaning. We evaluated a color segmentation method but the computational cost of preprocessing a single image was too high (from 15 seconds to a minute depending on the image), therefore the recommender system would not be able to give a result in an acceptable time.

1.3 Challenges and techniques

In this work we have chosen a content-based approach but, instead of using textual data, we work with visual content extracted from images in order to recommend tags. We considered MPEG-7 and SIFT like descriptors which are both visual features. We tried to obtain an optimal combination of these features in order to recommend tags accurately. We wanted to establish a relation between visual features and tags.

We chose to use SIFT like features because they are currently the best performing in the object recognition research area. With the use of these ad esdescriptors, we expected to identify representative places such as famous buildings or squares.

We also aimed at building a large scale application in order to train our recommendation engine because many of the proposed articles are limited to a few thousands images. Hence, we had to find scalable methods in order to train our recomender system and to compare the new images with the training set. This is especially challenging when we use SIFT features because the matching process has a high computational cost.

The final application should propose correctly ten tags per image in a limited time. We consider that if the tag recommendation takes more than ten seconds the experience for the user will be frustrating and will make the recommender useless. This really limited us because we had to discard many image processing algorithms who might have helped our pourpose but that implied a high computational cost.

For the achievement of our experiments, we worked with the Cophir dataset, which is a large collection of images (over 100 million) taken from Flickr. This dataset contains also some other information about the image posted by users (tags, views, comments...) and metadata like MPEG-7 features. This extra information makes the dataset suitable for content-based image retrieval research.
We observed that not all the images are equally tagged, which means that not all the images help equally studying the relation between tags and image features. After some hand-made analysis on a sample of the collection, we also realized that images with few tags would not be representative while images with too many tags would create ambiguity.
We also noticed the fact that not all the tags would be equally useful, as many of them appear only once or twice. On the other hand, we observed that the tags with more appearances such as 'wedding' or '2006' would not be significative. In fact, the tag appearance distribution over the dataset considered follows a power law as we can see in the next figure.

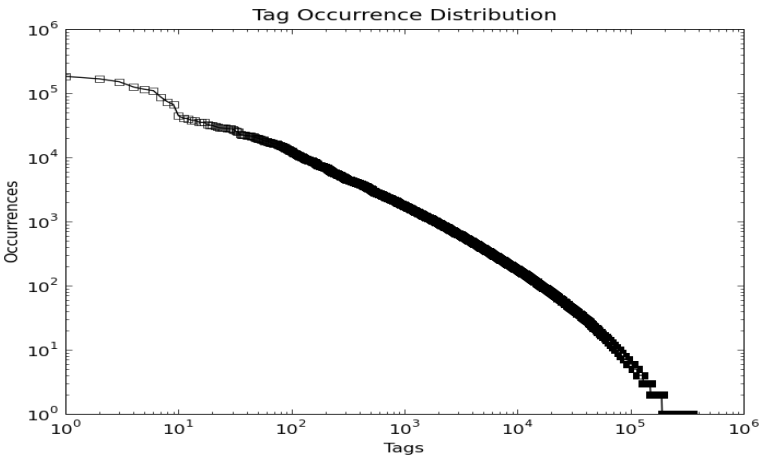


Figure 1.2: Tag appearance distribution

Therefore, we needed to filter both the images and the tags in order to avoid an excessive noise in the dataset. In fact, the noise in the dataset, specially the sparsity of tags is one of the biggest difficulties we found while trying to achieve our pourpose. This is mainly the reason why many of the articles that have been published reduce the tag recommendation problem to a classification problem.

1.4 Contribution of this thesis

In this thesis, we propose a scalable tag recommender based on visual features of the images. We use Mpeg-7 and SURF descriptors which are global and local features respectively. We present a combination of these features in order to recommend tags. For the final application, a set of 100000 images of the Cophir dataset is used in order to train the recommender.
The novel contributions of this thesis are as follows:

∗ In order to solve the scalability problem, we adopt a visual clustering technique. This splits the original collections in small sets that can be more easily processed.

∗ In particular, these images are firstly divided in textual clusters with the software k-metis. This first segmentation of the dataset is based on the similarity between the tags of the different images. This textual clustering helps in gathering together related images. The second segmentation is performed with a k-means algorithm based on the Mpeg-7 descriptors. This visual clustering helps in gathering together images with similar colors, textures and edges. The combination of both textual and visual clustering IS QUITE INNOVATIVE AND HELPS TO CREATE CLUSTERS OF IMAGES HAVING A STRONG SEMANTIC RELATEDNESS.

∗ ALSO WE EXPLOIT A SIGNIFICANT CLEANING OF THE DISCOVERED CLUSTERS. We identify the cases in which there is a strong relation between visual clusters and tags and rule out those clusters in which this connection does not exist. With this cleaning, we increase the efficacy of our recommender because the dataset we work with is really noisy but we also improve the efficiency.

∗ RELATED WITH THE SCALABILITY PROBLEM, the SURF features are also filtered in order to reduce the amount of keypoints in the dataset. In this case we also rule out the features that are not related to any tag. As the SURF matching process takes too long, this matching can only be computed with a reduced set of images. We present some new scalable techniques that allow us to select the best candidates among the whole dataset.

∗ REGARDING THE EFFICIENCY, we use of the M-tree structure that allows us to perform similarity searchs in an efficient way in our filtered dataset of both Mpeg-7 and SURF features.

∗ WE EVALUATED ESPERIMENTALLY THE QUALITY OF THE RECOMMENDED TAGS, BY DEFINING SOME METRICS ... We also propose some metrics in order to evaluate the accuracy of our recommender and to compare it with other techniques.

IN CONCLUSION,
With the aim of recommending tags for a given image, we follow an innovative strategy that combines the use of SURF features and the visual clusters obtained with the textual+visual clustering.

In figure 1.3 we show some examples of tags proposed by our recommender:

RECOMMENDED TAGS:
bluesky
woods
hiking
nature
france
summer
fire
sun
sky
winter

RECOMMENDED TAGS:
rome
italy
square
roma
campidoglio
piazza
italia
piazzanavona
photo
europe

RECOMMENDED TAGS:
morocco
mosque
marrakech
tower
koutoubia
maroc
minaret
marrakesh
africa
islam

RECOMMENDED TAGS:
paris
notredame
france
cathedral
faade
ledelacit
church
architecture
notredamedeparis
gothic

Figure 1.3: Four examples of tag recommendation

Chapter 2

Similarity matching based on MPEG-7 features

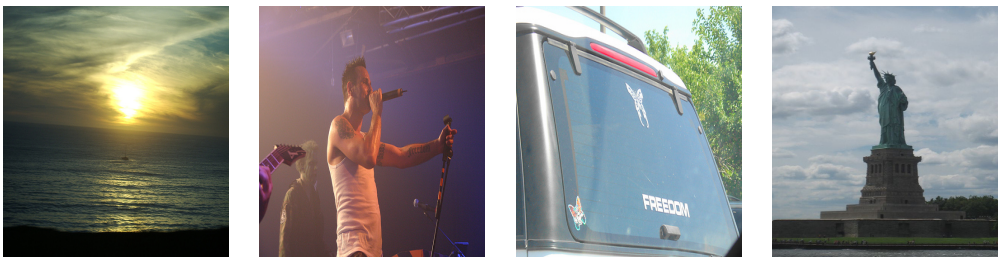
MPEG-7 features are global features that contain information concerning colors, textures and shapes of an entire image. These features have been widely used in content-based image retrieval. In fact, a search engine called Sapir has been developped using the MPEG-7 features avalaible in the CoPhIR dataset. In the following figure we can see an impressive result returned by Sapir. This similarity matching has been computed only exploiting the MPEG-7 descriptors.

Query image *similar images returned by Sapir*

Figure 2.1: Similarity search with Sapir

Nevertheless, the efficacy of techniques based on global features is not always satisfactory. This is due to the fact that it is hard to extract semantical or conceptual meaning from an image. In fact, a concept is hardly connected to features like colors, edges or textures. In order to illustrate that difficulty, we show the images returned by Sapir given the textual query 'freedom'. We can see that all the images returned are related to the term freedom despite the obvious differences between them.

Query: 'FREEDOM'



Images returned by Sapir

Figure 2.2: Textual search with Sapir

Another interesting fact to analyze is that MPEG-7 descriptors are good to describe panorama like photos, but fail to recognize concrete buildings.

Query image *similar images returned by Sapir*

Figure 2.3: Similarity search with Sapir

Given an image to analyse, our goal is to retrieve the images with the nearest descriptors in a dataset. This implies a preprocessing of the dataset (feature extraction) and the use of a metric for each descriptor. In fact, as we work with six different MPEG-7 descriptors, we can think of a dataset as six independent metric spaces, each one corresponding to a feature. In these metric spaces, the distances defined depend on the descriptor and the searchs can be accomplished in an efficient way.

In this section, we explain the feature represented by each of the MPEG-7 descriptors that we have used. We explain how the distance between two descriptors is computed and we show some visual examples of similarity searchs in a reduced dataset. For each descriptor, we specify whether we have used the standard MPEG-7 distance measure or an improved one. In the second case we specify the work in which it has been proposed.

2.1 Scalable color type descriptor

The Scalable Color Descriptor is a color histogram quantized in a HSV (Hue,Saturation,Value) color space. We compute the distance between an image A and an image B as follows:

Let the descriptors be:

$$f_A = \{coeffs_A[64]\}$$
$$f_B = \{coeffs_B[64]\} \tag{2.1}$$

We compute the distance between an image A and an image B as follows:

$$D(A,B) = \sum_{i=0}^{63} |coeffs_A[i] - coeffs_B[i]|$$

This metric corresponds to the standard MPEG-7 distance measure.

Query image *Nearest images*

Figure 2.4: Similarity search based on the Scalable Color Descriptor

In this figure, we show two examples of similarity searchs in a reduced dataset. The first query image shows a landscape while the second is a photo of the Empire State Building. We can see that the query image and the returned images have similar colors. In this case, we are only compairing colors, therefore it is normal that query image and retrieved images have a different semantic meaning.

2.2 Color layout type descriptor

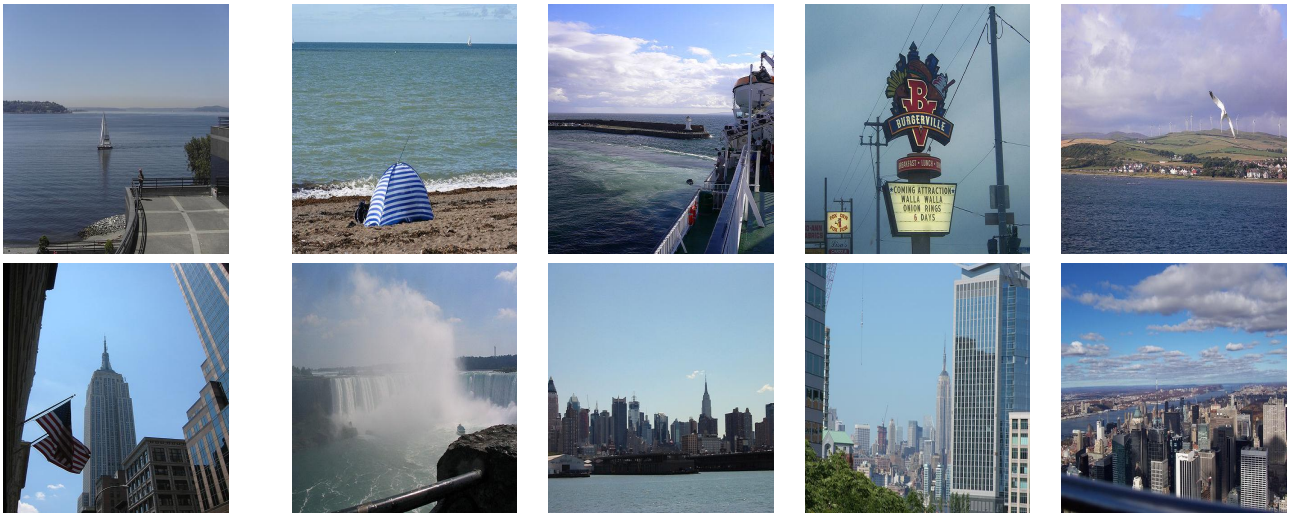
The Color Layout Descriptor divides the image in a 8x8 grid, and for each of the subimages the representative color is given. Let the descriptors be:

$$f_A = \{YDCCoef_A, CbDCCoef_A, CrDCCoef_A, YACCoef_A[5], CbACCoef_A[2], CrACCoef_A[2]\}$$
$$f_B = \{YDCCoef_B, CbDCCoef_B, CrDCCoef_B, YACCoef_B[5], CbACCoef_B[2], CrACCoef_B[2]\}$$

We compute the distance between an image A and an image B as follows:

$$\begin{aligned} D(A,B) = & \quad |YDCCoef_A - YDCCoef_B| \\ + & \quad |CbDCCoef_A - CbDCCoef_B| \\ + & \quad \sum_{i=0}^{4} |YACCoef_A - YACCoef_B| \\ + & \quad \sum_{i=0}^{1} |CbACCoef_A - CbACCoef_B| \\ + & \quad \sum_{i=0}^{1} |CrACCoef_A - CrACCoef_B| \end{aligned}$$

This metric corresponds to the standard MPEG-7 distance measure.



Query image *Nearest images*

Figure 2.5: Similarity search based on the Color Layout Descriptor

In this figure, we show the results of two similarity searchs performed with the Color Layout Descriptor. It is important to note that we have used the same query images that we used in the previous example. This way we can compare and understand the feature represented by each descriptor. In this case, this descriptor represents the image as a 8x8 grid and represents the most important colors of each grid. Therefore, this descriptor is suitable for the retrieval of panorama photos or landscapes.

2.3 Color structure type descriptor

The Color Structure Descriptor divides the image in a 8x8 grid, and for each subimage it describes the distribution of the different colors. Therefore it represents local color features of the images. In order to compare two images A and B we proceed as follows:

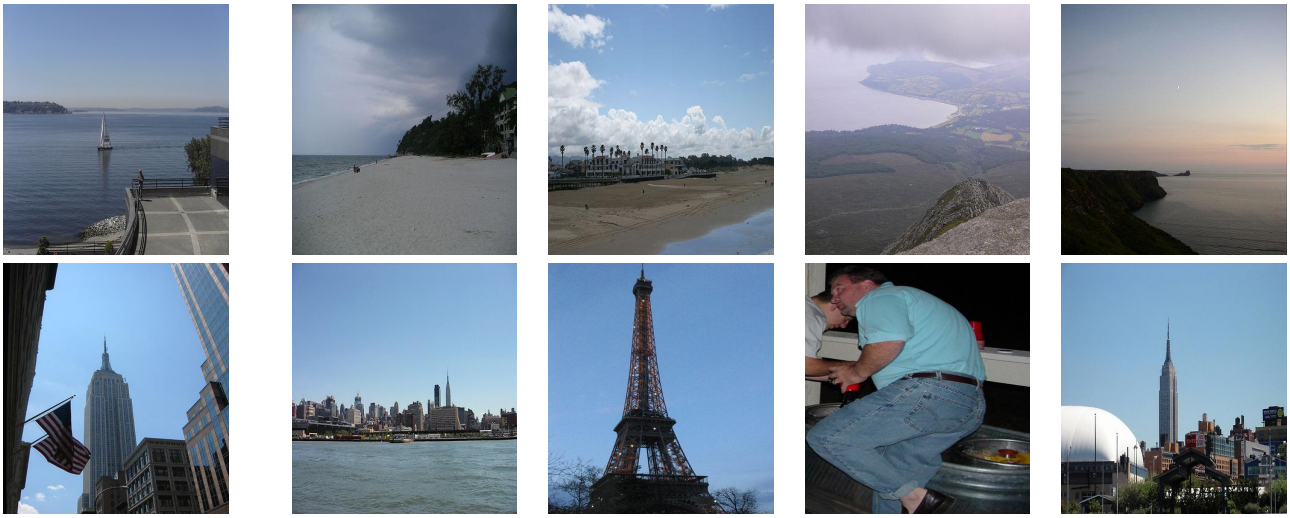
Let the descriptors be:

$$f_A = \{values_A[64]\}$$
$$f_B = \{values_B[64]\}$$

We compute the distance as follows:

$$D(A,B) = \sum_{i=0}^{63} |values_A[i] - values_B[i]|$$

This metric corresponds to the standard MPEG-7 distance measure.



Query image *Nearest images*

Figure 2.6: Similarity search based on the Color Structure Descriptor

In this figure, we show the results of two similarity searchs performed with the Color Structure Descriptor. We have used the same query images that we used in the previous examples. In this case, this descriptor represents the image as a 8x8 grid and represents the color distribution of each grid. This descriptor is similar to the Color Layout Descriptor and is also suitable for the retrieval of panorama photos or landscapes. Once again, we observe that MPEG-7 descriptors fail to recognize concrete buildings, in fact an image of the Eiffel Tower is retrieved while the query image represented the Empire State Building.

2.4 Dominant color type descriptor

The Dominant Color Descriptor is a compact descriptor that contains a small number of representative colors. These colors are obtained by clustering and color quantization. The feature vector is composed by the representative colors and their percentages in a region, the spatial coherency of the colors and the color variances.

Let N_1 and N_2 be the number of dominant colors present in A and B repectively. Then the descriptors are represented as follows:

$$f_A = \{(c_{a_{i1}}, c_{a_{i2}}, c_{a_{i3}}), p_{a_i}, s_a, (v_{a_{i1}}, v_{a_{i2}}, v_{a_{i3}})\} \quad with \quad i = 0 \rightarrow N_1$$
$$f_B = \{(c_{b_{i1}}, c_{b_{i2}}, c_{b_{i3}}), p_{b_i}, s_b, (v_{b_{i1}}, v_{b_{i2}}, v_{b_{i3}})\} \quad with \quad i = 0 \rightarrow N_2$$

In this case we decide not to use the standard MPEG-7 distance measure. In order to compute the distance between two images A and B, we adopt the technique of the merging palette proposed in [3]. As explained in this article, histogram intersection cannot be applied while matching with the DCD because the space of each descriptor is not the same. It is important to note that the properties of the metric spaces are respected with the use of this distance function.

Query image *Nearest images*

Figure 2.7: Similarity search based on the Dominant Color Descriptor

In this figure, we show the results of two similarity searchs performed with the Dominant Color Descriptor. We have used the same query images that we used in the previous examples. In this case, this descriptor represents the most important colors of the image and the percentage of the image in which they appear. This descriptor only focuses on a reduced number of colors per image, therfore it is not as performant as other descriptors in retrieval of panorama photos.

2.5 Homogeneous texture type descriptor

The Homogeneous Texture Descriptor consists in the mean, the standard deviation, the energy and energy deviation values of an image. Information of texture is extracted from the frequency domain and the values are frequency coefficients. The frequency domain is divided into 30 channels using filters with sensitivity of 5 differents scales and 6 different orientations.

Let the descriptors be:

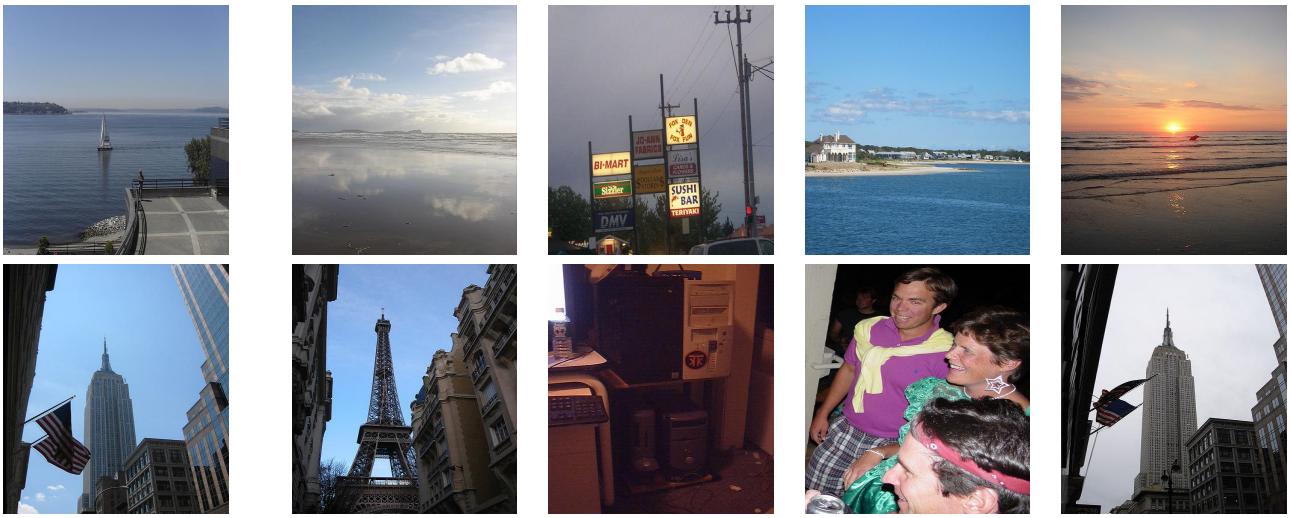
$$f_A = \{average_A, stddeviation_A, energy_A[30], energydev_A[30]\}$$
$$f_B = \{average_B, stddeviation_B, energy_B[30], energydev_B[30]\}$$

We compute the distance between two images A e B as follows:

$$s_1 = \frac{(stdDeviation_A + stdDeviation_B)}{2}$$

$$D_2(A,B) = \frac{\sum_{i=0}^{29}(|energyA[i] - energyB[i]| + |energyDevA[i] - energyDevB[i]|)}{s_1}$$

This metric corresponds to the standard MPEG-7 distance measure.



Query image *Nearest images*

Figure 2.8: Similarity search based on the Homogeneous Texture Descriptor

In this figure, we show the results of two similarity searchs performed with the Homogeneous Texture Descriptor. We have used the same query images that we used in the previous examples. In this case, this descriptor identifies different areas and shapes of the image. In fact, these shapes are the limit between two areas with different texture. This descriptor does not focus on colors, therefore it is interesting to combine this descriptor with the previous ones.

2.6 Edge histogram type descriptor

The Edge Histogram Descriptor divides the image in a 4x4 grid, and for each of the subimages it describes the distribution of five different edges: vertical, horizontal, 45-degree diagonal, 135-degree diagonal and non-directional. The histogram for each subimage represents the relative frequency of these edges. Hence, 5 bins are required to describe a subimage and 4x4x5=80 to describe an image. In the article [2], Chee Sun Won, Dong Kwon Park and Soo-Jun Park show that this descriptor is more efficient with information about global and semiglobal edges in the image and that this information can be computed with the 80-values descriptor.

A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)
A(2,0)	A(2,1)	A(2,2)	A(2,3)
A(3,0)	A(3,1)	A(3,2)	A(3,3)

Figure 2.9: 4x4 grid

In this case we decide not to use the standard MPEG-7 distance measure.
In order to compare two images A and B we compute the distance D(A,B) with an approach based on [2]. Let the descriptors be:

$$f_A = BinCounts_A[80]$$
$$f_B = BinCounts_B[80]$$

We compute the distance between two images A e B as follows:

$$\begin{aligned} D(A,B) = & \sum_{k=0}^{79} |BinCounts_A(k) - BinCounts_B(k)| \\ + & \quad 5 * \sum_{k=0}^{4} |Global_A(k) - Global_B(k)| \\ + & \sum_{k=0}^{49} |SemiGlobal_A(k) - SemiGlobal_B(k)| \end{aligned}$$

where:

- $BinCounts_A$ and $BinCounts_B$ are the feature vectors used before,

- $Global_A$ and $Global_B$ represent the normalized histograms of the global edges of images A and B

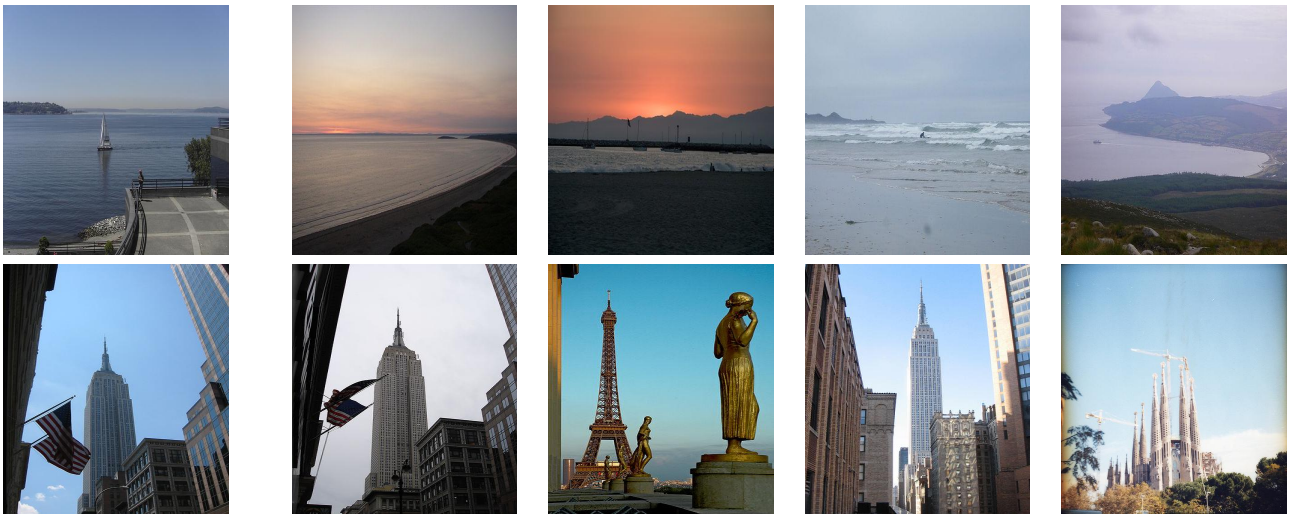
- $SemiGlobal_A$ and $SemiGlobal_B$ represent the normalized edge histograms of the 10 following segments of images A and B:

  - the rows: A(i,0) A(i,1) A(i,2) A(i,3), i=0 $\rightarrow$ 3 (4 segments)

  - the columns: A(0,j) A(1,j) A(2,j) A(3,j) j=0 $\rightarrow$ 3 (4 segments)

  - the diagonal A(i,i) i=0 $\rightarrow$ 3 (1 segment)

  - and the diagonal A(0,3) A(1,2) A(2,1) A(3,0) (1 segment)



Query image *Nearest images*

Figure 2.10: Similarity search based on the Edge Histogram Descriptor

In this figure, we show the results of two similarity searchs performed with the Edge Histogram Descriptor. We have used the same query images that we used in the previous examples. In this case, this descriptor divides the image in an 8x8 grid and identifies the egdes in these subimages. Therefore, this descriptor is suitable for the retrieval of images containing landscapes but also for images containing buildings or squares.

Chapter 3

Similarity matching based on SURF keypoints

Currently, a lot of research is being made in object recognition. One of the most popular tools are the Scale Invariant Feature Transform normally refered as SIFT features. They were introduced by David G. Lowe in 1999. For example, the tool Goggles proposed by Google is supposed to use SIFT keypoints.

3.1 Scale Invariant Feature Transform

SIFT features are local features used to recognize places, buildings and objects. The usual procedure to compare two images A and B is described in the following:po

3.1.1 SIFT features extraction

We are not going to explain this procedure in detail because we consider it is not our goal but we illustrate each step of the procedure using the explanation given by Ballard Blair and Chris Murphy in [x] and Dan Huttenlocher in [x]. Given two images of the same scene with a scale difference between them, the goal is to find the same keypoints independently in each image. The two images to analayze must be processed independently, i.e. we first have to extract the keypoints of each image and confront them only in the matching procedure. Lowe proposed to look for the points that maximize the Difference of Gaussians over scale and space. For each image, we obtain the Difference of Gaussian that can be conceived as a 'pyramid' of several images, each being a unique difference of Gaussians. The procedure to build this pyramid is the following:

step 1: The image is blurred repeatedly: the difference between two consecutive blurs is one 'Octave' of the pyramid.
step 2: One of the blurred images is downsampled by a factor of two, and the process is repeated producing an output in a different size.

We show here an example of this process. We can see two blurred images and their difference of Gaussians.

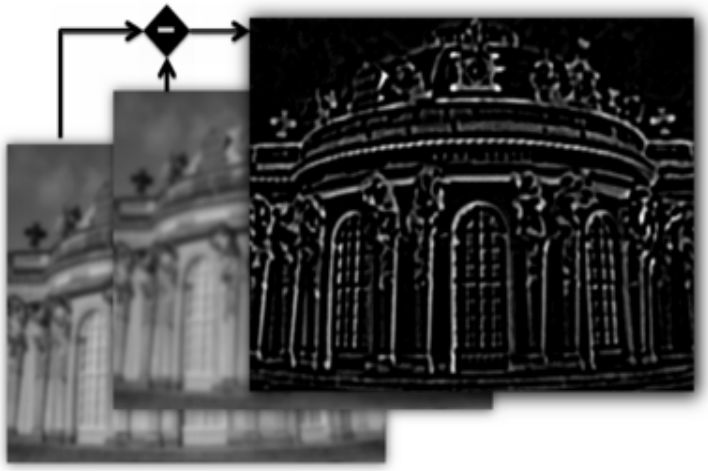


Figure 3.1: Example of the Difference of Gaussians between two blurred images

We show here an example of the whole process of constructing this pyramid:

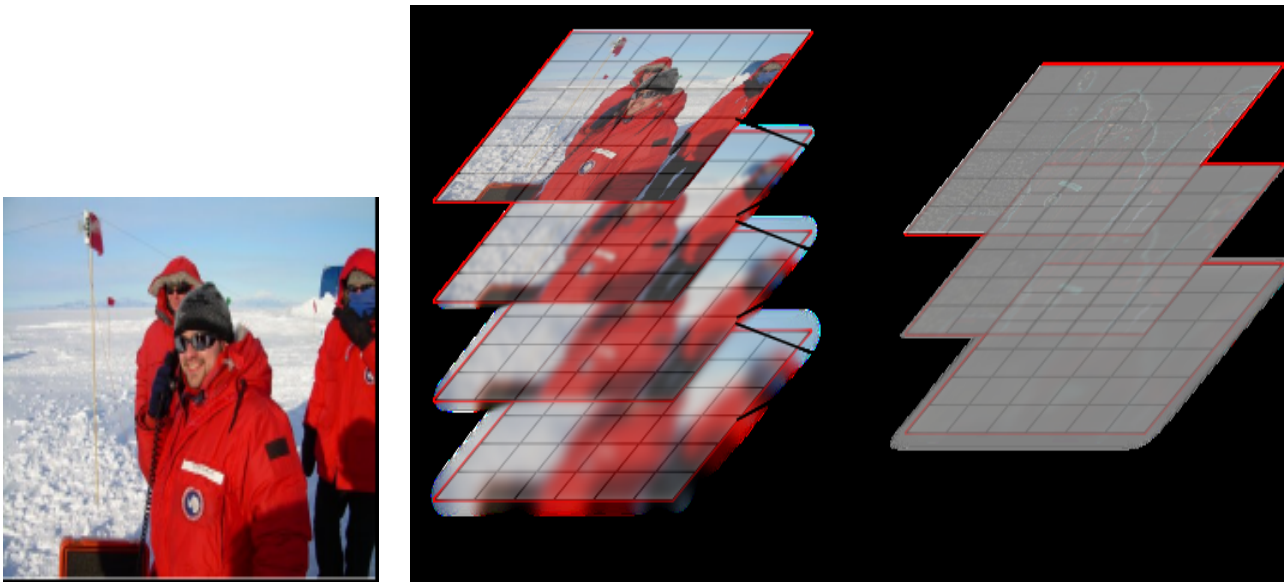
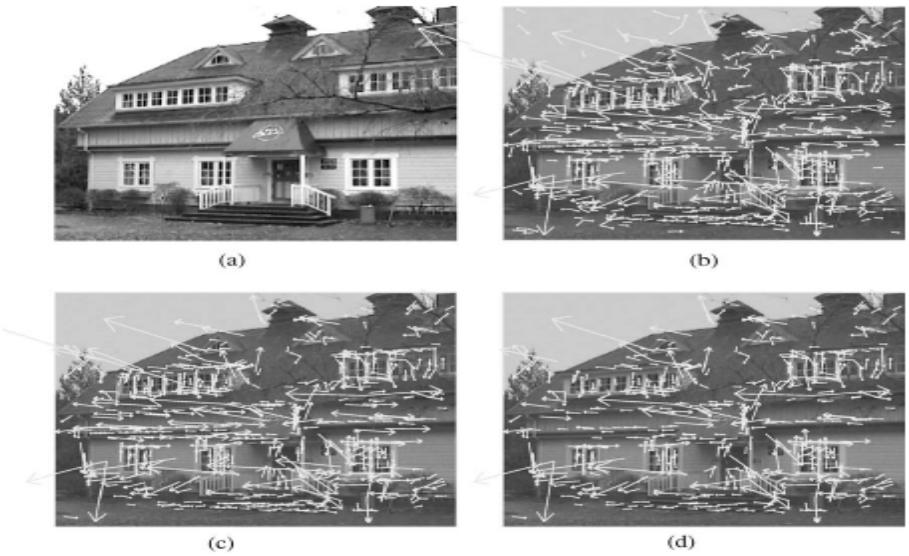


Image to process Pyramid of Differences of Gaussians

Figure 3.2: Process of constructing the pyramid of Differences of Gaussians

Now that we have obtained this pyramid, we look for the pixels in the first level who have high contrast with pixels that surround them. This contrast must occur for that same pixel in all the levels of the pyramid. Generally, a big a set of features is obtained. These features must be filtered because not all of the points are equally interesting. Two filters are applied: removing features with low contrast and removing features with low curvature. The images below are an example of this filters:



(a) Image to process (b) All features found
(c) Low contrast removed (d) Low curvature removed

Figure 3.3: Filters applied to the features found

3.1.2 Matching of the SIFT feature vectors

The author David G. Lowe has released a demo software who implements these two steps. The input for the matching program are two keys, one for each image, containing the features extracted as explained before.

Let
$keys_A$ be the set of SIFT features of A
$keys_B$ be the set of SIFT features of B

For each of the features k_i conforming the set $keys_A$, we look for the 2 closest features in $keys_B$. If we find two features with a high difference between them, we consider that we have found a match. On the other hand, if the two features found in $keys_B$ have little difference between them, we discard the keypoint k_i from $keys_A$. This filter is done because the situation in which two similar features are found in $keys_B$ usually correspond to false matches due to the noise in the images. As we can see in the algorithm showed below, a parameter α is introduced in order to perform this filter.

The matching process follows the next algorithm:

for $k_i \in keys_A$ **do**
  look for the 2 closest $k_1, k_2 \in keys_B$
  if $d(k_1, k_i) < \alpha * d(k_2, k_i)$ **then**
    $match++$
  end if
end for

We observe that the matching process depends on the order of the input: the number of matches found when we compare $keys_A$ with $keys_B$ is different that the amount of matches produced by comparing $keys_B$ with $keys_A$.

We now show a matching example proposed by Lowe.

Figure 3.4: Example proposed by Lowe

3.2 SIFT vs SURF

There are other similar descriptors inspired in SIFT keypoints such as Fast Approximated SIFT, Color SIFT and SURF(speeded up robust features). These other methods are supposed to perform similarly. We are specially interested in the case of SURF keypoints because the extraction step is supposed to be much faster. This difference in time might be of a great relevance to us because we pretend to build a tag recommending system with a fast response time.

In this section we compare the efficacy of both descriptors. As our first contact with this kind of features, we perform a little experiment. We try to discover the cases in which those features are useful. We choose the four following 4 couples of images:

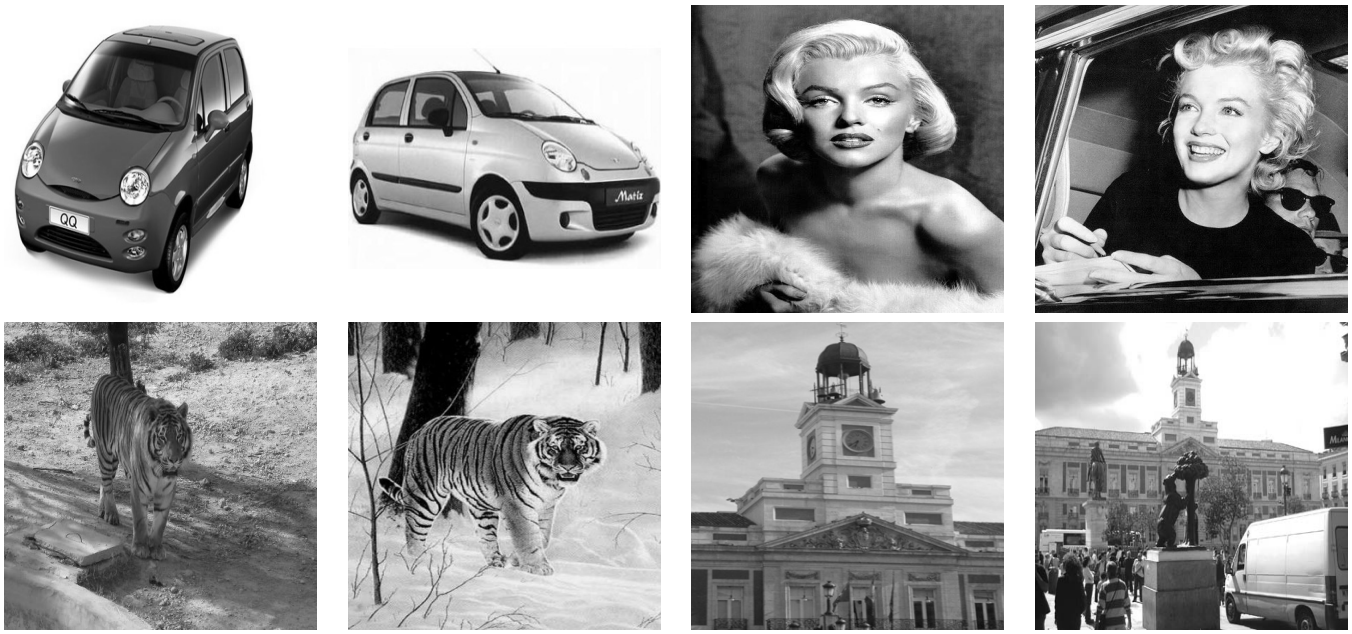
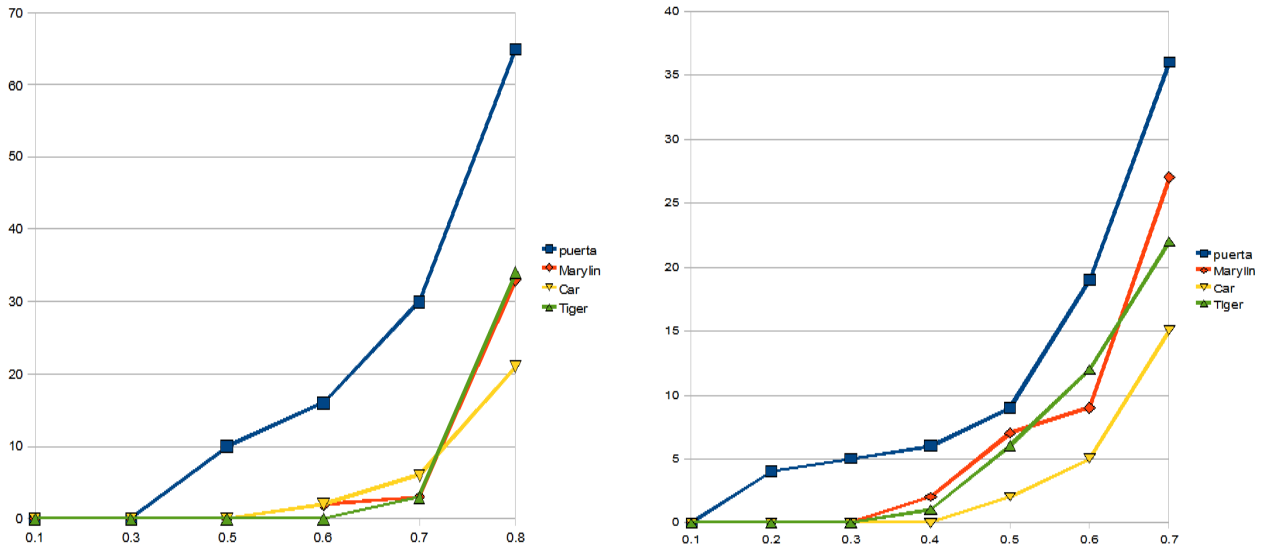


Figure 3.5: Four couples of images used in this first approach

As we mentionned before, the matching technique uses a parameter α who determines whether a match has to be considered noise or not. We try the matching technique proposed for the different couple of images and we count the amount of matches found. We expose the results obtained with SIFT and SURF features.



SIFT recommended alpha=0.6

SURF recommended alpha=0.5

Figure 3.6: Matches found with SIFT and SURF descriptors

As we can see, SIFT features work great only in the case of the images of Puerta del Sol. SURF features seem to also make a positive match for the images of Marylin and the tigers. But when we looked in detail, we realized that the matches found with SURF features in the cases of Marylin, the cars and the tigers corresponded to wrong matches.

We can see that keypoints found correspond to points a with high color contrast such as corners or edges. We can also see that some of the matchings found are wrong matches. In fact, most of the matches found between the others couples of images correspond to false matches. Therefore, we have to find different ways to compare two images because counting the amount of matches seems not to be reliable enough. We also need to find more efficient ways to proceed with the matching because this method could not be used for large datasets.

Figure 3.7: Matches found with SURF descriptors

We can conclude that, even though both features can be used to recognize places, SIFT are more powerful than SURF features. Nevertheless, SURF features extraction is way more efficient. If we are to include this kind of descriptors in a real time application we are forced to choose SURF features. We also need to find alternative and more robust ways to match images.

3.3 Distances and metrics used with SURF features

Theoretically, two images taken from the same place should have many matches. We have discovered that the number of keypoints found in an image depends on many factors like resolution, brightness etc. Therefore, good quality images are largely better to tell whether two images match or they don't. This represents a great difficulty for us because many of the Flickr photos that we are working with are fuzzy or taken with a low resolution camera.

We now show the data format of SURF keypoints. Let k be a SURF keypoint, then:

$$k = \{x, y, a, b, c, l, des\}$$

where

$$\begin{aligned} x, y &= \text{position of interst point} \\ a, b, c &= \text{entries of second moment matrix} \\ l &= \text{sign of laplacian (1 or -1)} \\ des[64] &= \text{descriptor vector} \end{aligned}$$

Two SURF keypoints k_1 and k_2 are comparable if they have the same sign of laplacian. In this case, the distance between them is the euclidean distance between their descriptor vector.

We show here how the distance between two keypoints k_1 and k_2 is computed:

$d = \infty$
if $l_1 = l_2$ **then**
  d=0
  for $i = 0; i < 64; i++$ **do**
    $d = d + (des_1[i] - des_2[i])^2$
  end for
  $d = \sqrt{d}$
end if
return d

It is important to remember that the amount of matches between two images A and B depends on the order of the matching process: matching A with B and matching B with A return different results. During our first experiments, we observed two interesting facts:

observation 1: counting the number of bimatches (keypoints who match in both orders) is a really strong similarity measure

observation 2: when a match takes place, we identify a relation between distance of the two keypoints and the accuracy of the match: the closer two keypoints are, the higher are the chances of this match to be accurate.

In order to compare two images A and B, we propose different similarity measures based on this two observations:

3.3.1 Search of the nearest match

This metric is based on the relation observed between distance and accuracy of the matches. We find all the matches between the images A and B as explained in section 3.1.2. We quantify the distance between two keypoints that form a match computing the euclidean distance between their descriptor vector. We consider the nearest match between A and B to be the match with the minimum distance. From now on we call this distance D_1.

We proceed as follows:

$D_1 = \infty$
for $k_i \in keys_A$ **do**
  look for the 2 closest $k_1, k_2 \in keys_B$
  if $d(k_1, k_i) < \alpha * d(k_2, k_i)$ **then** {THIS INDICATES IF A MATCH TAKES PLACE}
    if $d(k_1, k_i) < D_1$ **then**
      $D_1 = d(k_1, k_i)$
    end if
  end if
end for
return D_1

3.3.2 Search of the n nearest matches

This metric is also based on the relation observed between distance and accuracy of the matches and consists in an extension of the precendent method. Given the images A and B, we now look for the n nearest matches between A and B.
Let d_i be the distance of the i-th nearest match between A and B.
Then $D_n = \sum_{i=1}^{n} d_i$
From now on we call this distance D_n (we use D_3 and D_5).

3.3.3 Amount of matches

This method is the one proposed by the authors of both SIFT and SURF features. Given the images A and B we compute the number of matches m between the features of A and the features of B. The matches are computed as in the software proposed by Herbert Bay, Andreas Ess, Tinne Tuytelaars and Luc Van Gool (authors of [6]):

$$matches(A, B) = \#k_i \in keys_A \text{ that match with any } k_j \in keys_B$$

3.3.4 Amount of bimatches

This measure is an extension of the previous one. As explained before, the matches obtained with the standard matching method depend on the order of the input. In this case, we consider those keypoints who match in only one sense as noise. Therefore we count a bimatch only when a feature f_i of A matches with a given feature f_j of B when we match the images A and B in both senses:

$bimatches(A, B) = \#(k_i, k_j), k_i \in keys_A$ and $k_j \in keys_B$ such that $match(k_i, k_j)$ and $match(k_j, k_i)$

Chapter 4

Algorithms proposed

4.1 Cleaning of the dataset

As the dataset considered is huge, we need to work with a reduced subset of images. Initially, we decided to work with 5 million images taken randomly from the whole Cophir dataset.

We observed that many images with few tags had been automatically labeled with numbers, dates, camera models etc... We concluded that images with 2 or less tags would not be useful to establish a relation between tags and image features. On the other hand, when images are tagged with more than 20 tags, these tags tend to be vague. This situation usually corresponds to images uploaded as an album with the same tags for each of the images. These images would introduce ambiguity and noise in the dataset. Therefore, as a first filtering, we kept only the images containing more than 2 and less than 20 tags.

At this point the set of images remained huge (2280668 images) and really noisy so more filtering was needed. The tags proposed by users are in different languages, many are abbreviations like 'bw', 'mr', or 'dr'(black and white, mister or doctor). In addition to that, many of them are bad spelled. We also find that many tags are the result of the union of two or more words words such as ('empirestatebuilding') or even phrases('dontknowwhat'). Fortunately, most of these tags are easy to eliminate because they appear only once or twice in the dataset. The opposite situation, i.e tags with a really high number of appeareances as '2006', '2007' or 'holiday' also have to be eliminated because they are hardly connected to image features. We obviously wanted to get rid of all these tags that would not help our pourpose and at the same time we wanted to reduce the number of tags to work with. Hence, as a second filtering, we did not to consider the tags with less than 3 and more than 40000 occurrences.

In the next sections we explain all the steps we followed until we reached an acceptable performance recommending tags. As exposed in the previous sections, our image searchs are based on MPEG-7 descriptors and SURF keypoints. We observed that MPEG-7 descriptors are suitable for content-based image retrieval in the case of landscapes or nature related photos. Whereas SURF keypoints should be able to recognize concrete places or buildings. We need datasets who allow us to proove this difference in their respectives performances, therefore we decide to work with two reduced independent datasets.

The first dataset (DAT1) contains mainly images of representative buildings, this dataset should help us proove the efficacy of SURF keypoints. In order to obtain this dataset, we select 2000 images containing one of the following tags: 'sagradafamilia', 'empirestatebuilding', 'toureiffel', 'colosseo', 'trevifountain', 'puertadelsol' and 'puertadealcala'. We also include other images picked at random.

The second dataset (DAT2) is more suitable for MPEG-7 descriptors, hence we choose images with one or more of the next tags: 'fire', 'bluesky', 'brick', 'sea', 'woods' and 'sand'.

We now show the distribution of these images in their respectives datasets.

Dataset DAT1:

- 500 images tagged with "sagradafamilia"
- 448 images tagged with "empirestatebuilding"
- 423 images tagged with "toureiffel"
- 116 images tagged with "colosseo"
- 31 images tagged with "trevifountain"
- 6 images tagged with "puertadelsol"
- 4 images tagged with "puertadealcala"
- 478 random images

Dataset DAT2:

- 250 images tagged with "fire"
- 250 images tagged with "bluesky"
- 250 images tagged with "brick"
- 250 images tagged with "sea"
- 250 images tagged with "woods"
- 250 images tagged with "sand"

4.2 Metrics

From now on, we measure the efficacy of the tags reccomended as follows. The two datasets are divided in training (2/3 of the dataset) and validation set (1/3 of the dataset). Therefore, for the first dataset there are 1333 images in the training set and 667 in the validation set while for the second dataset 1000 images form the training set and 500 images form the validation set.

4.2.1 Tagging accuracy

For each of the images in the validation set we recommend a maximum of 10 tags. In other papers, tags recommended are classified into categories such as very good, good, bad, very bad and don't know. We believe this evaluation is subjective. It also implies a manual evaluation. Therefore we initially try to avoid this kind of measurement. As the images in the validation sets are in the Cophir dataset, we know all their original tags. We consider a recommended tag to be good if it is contained in the original tags. In the results of our experiments we show a percentage that corresponds to the percentage of original tags recommended. For example, in the case an image had 4 original tags, if we recommended 2 of these original tags we count as 50% of tagging accuracy while if we recommended the 4 of them we count it as 100% of tagging accuracy. Note that if an image had 15 original tags and we recommended 10 of these tags we would still count it as a 100% of tagging accuracy. This measure is suitable for our experiments because it does not imply a manual evaluation of the recommended tags which, other that being an objective measure, makes us save a lot of time. Nevertheless we are aware of the fact that images are not usually well tagged, which means that this evaluation method is far from being optimal. In fact, a recommended tag could actually fit perfectly with a given image and still be considered as a failure whereas a tag considered as a success might not be really suitable for the image.

4.2.2 Classification accuracy

The images forming the two datasets have been chosen because they are labeled with specific tags. Hence, we can consider that these images are divided into different categories depending on whether or not they have one of these tags. We call these tags 'category tags'. The evaluation of the classification is specially interesting in the first dataset, where identifying a concrete building is the main objective. We propose two different measures, we see if a 'category tag' has been recommended (SAID CLASSIFICATION) and if it has been recommended before any other 'category tag' (CLASSIFICATION). In the case of the second dataset, this measurement is not really relevant because these categories are not disjoint. In other words, tags like 'sea', 'sand' and 'bluesky' may perfectly happen to be suitable for a given image. When an image is tagged with more than one 'category tag', we consider that its category is the first 'categorty tag' to appear. Therefore, we simplify the problem and consider that an image can only belong to one category.

4.3 First approach

As a first approach, we work only with the MPEG-7 descriptors. Given an image to analyze, we perform a similarity search for each of these descriptors. We compute the distances as mentioned in section 2. As a result of the search, we obtain an image for each of these descriptors. We then get the tags from these 6 images (an image per descriptor). At this point, we have six different sets of tags, each set obtained with a different descriptor:

- $tags_{sc}$ formed by the tags of the image found with the Scalable Color descriptor
- $tags_{cl}$ formed by the tags of the image found with the Color Layout descriptor
- $tags_{cs}$ formed by the tags of the image found with the Color Structure descriptor
- $tags_{dc}$ formed by the tags of the image found with the Dominant Colors descriptor
- $tags_{ht}$ formed by the tags of the image found with the Homogeneous Texture descriptor
- $tags_{eh}$ formed by the tags of the image found with the Edge Histogram descriptor

The best case scenario is to find tags that appear in all of these sets, but it is very difficult due to the sparsity of tags in the dataset. In fact, the intersection of these sets is usually empty, hence we need to find a way to select tags other than the intersection. We observe that not all the descriptors behave equally, so it is necessary to weight the tags proposed by the different descriptors. We use the weights given in the article [Saphir] which are:

- Scalable Color: $w_{sc} = 2$
- Color Layout: $w_{cl} = 2$
- Color Structure: $w_{cs} = 3$
- Dominant Colors: $w_{dc} = 1$
- Homogeneous Texture: $w_{ht} = 0.5$
- Edge Histogram: $w_{eh} = 4$

We also work with the dominant color descriptor and its weight is assigned accordingly to the efficacy showed in our experiments.

Given the weights showed before, we give a score to each of the tags found as follows:

for each tag t found **do**
  $score(t) = 0$
  for $d \in \{sc, cl, cs, dc, ht, eh\}$ **do**
    if $t \in tags_d$ **then**
      $score(t) = score(t) + w_d$
    end if
  end for
end for

ALGORITHM 1

We call the tags with the highest scores the most voted tags. We adopt the following strategy in order to recommend tags:

for $d \in descriptors$ **do**

find the nearest image
end for
recommend the tags with the highest score

The results obtained with this method are not satisfactory, mostly because the recommended tags are usually too specific an don't fit well for the given images. We also realized that a great number of images in Flickr are uploaded as an album and have the same tags no matter what the visual content of the single image is. This means that for many images visual content and tags are barely related.

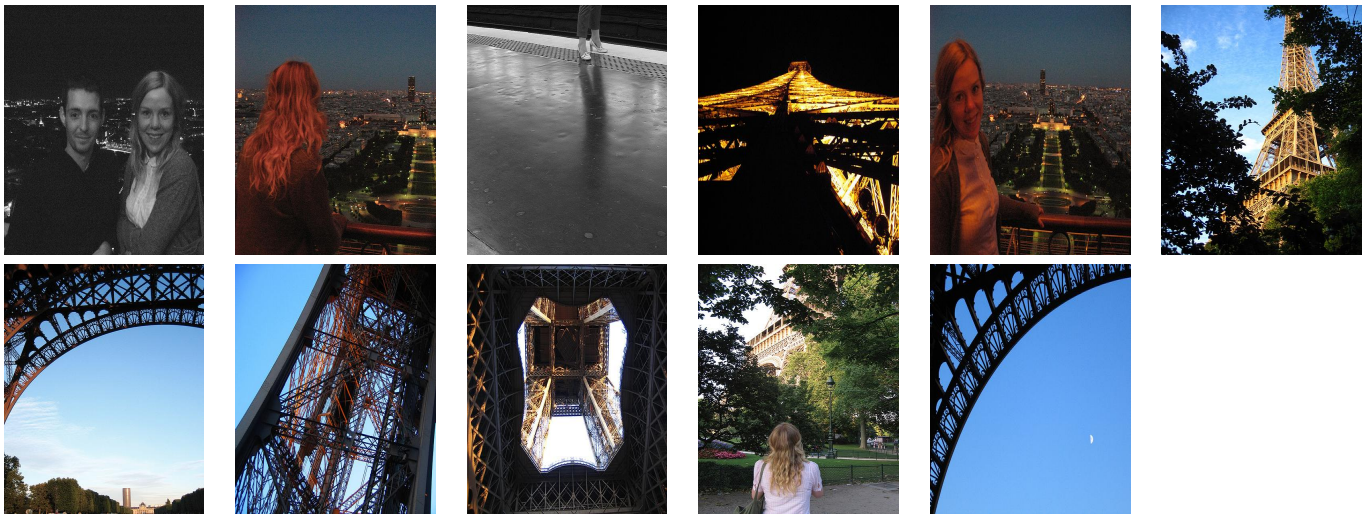


Figure 4.1: Example of an album uploaded by a same user

The figure 4.1 corresponds to an album uploaded by a same user. All the images are tagged equally. Their tags are 'toureiffel', 'paris' and 'july'. As we can see, the Eiffel tower appears in seven of the eleven images forming the album but most of them are partial views. Three of the photos are views from the Eiffel tower and the remaining one has definitively nothing to do with these tags.

ALGORITHM 2

We tried to solve these difficulties looking for the m nearest images for each descriptor. Given an image to analyze, we get the tags from these 6*m images (m images per descriptor). Therefore have six different sets of tags, each set obtained with a different descriptor:

- $tags_{sc}$ formed by the tags of the m images found with the Scalable Color descriptor
- $tags_{cl}$ formed by the tags of the m images found with the Color Layout descriptor
- $tags_{cs}$ formed by the tags of the m images found with the Color Structure descriptor
- $tags_{dc}$ formed by the tags of the m images found with the Dominant Colors descriptor
- $tags_{ht}$ formed by the tags of the m images found with the Homogeneous Texture descriptor
- $tags_{eh}$ formed by the tags of the m images found with the Edge Histogram descriptor

It is important to note that a same tag may appear more than once in a given set. We give a score to each of these tags as follows:

for each tag t found **do**

$score(t) = 0$
for $d \in \{sc, cl, cs, dc, ht, eh\}$ **do**
if $t \in tags_d$ **then**
$times(t, d) =$ occurrences of t in $tags_d$
$score(t) = score(t) + w_d * times(t, d)$
end if
end for
end for

for $d \in descriptors$ **do**
find the nearest m images
end for
recommend the tags with the highest score

We observed that the accuracy of the tags proposed increased significatively. Still, we did not reach the expected performance mainly because many uploaded albums contain quite identical images. Hence, many times the result of our similarity search is a set of m identical images with the same tags. So looking for more images does not always solve the problems explained before.

Figure 4.2: Example of an album containing quite identical images

4.4 Baseline

We decide to adopt a visual clustering strategy in order to solve some of the problems mentionned before. The main goal is to obtain a stronger relation between tags and visual content. For each descriptor, we proceed with a k-means clustering algorithm based on the MPEG-7 distances showed in chapter 2. If the clusters are big enough, the errors caused by uploaded albums should be reduced. With the introduction of clusters, we also increase the efficiency because we only compute the distances of a given image with the centroids of the clusters. Hence the amount of distances computed decreases significatively. This is important to us because one of the goals to achieve is to implement a big-scale application afterwards. We try to find an adequate number of clusters k for our datasets. We are working with two training sets of 1333 and 1000 images. We consider that each cluster should contain from 15 to 50 images. Therefore we experiment with k=25,50,75. We observe that the best clusters are usually those containing approx. 25 images. Clusters that are too big contain highly different images while small clusters usually contain identical images corresponding to albums. We consider appropiate to create clusters of approximatively 25 images, hence we choose k=50. Nevertheless the amount of images per cluster has a high variance due to the use of the k-means algorithm.

Recommendation Strategy

Our first experiment is based on the article 'Visual Language Modeling for Image Classification'. We adopt a recommending strategy based on the frequency of tags and the correlation between clusters and tags. Given a cluster C_i, the goal is to find the tags that are most strongly related to C_i . Therefore, we consider the global frequency of a tag and its probability to appear with the given cluster C_i. For each of the tags that cooccur with a given cluster C_i, we compute its score as follows:

$$score(t, C_i) = frequency(t) * correlation(t, C_i) \tag{4.1}$$

The higher are the global frequency of the tag and its correlation with a given cluster, the higher the final score is.

We define:

$vclusters(D_m) = \{C_1..C_k\}$ is the set of visual clusters found with a given MPEG-7 descriptofrequencyr D_m

$tags(C_j, D_m) = \{t_1..t_n\}$ is the set of tags occurring with at least one image of the visual cluster C_j

$$occurrences(t_i, D_m) = \sum_{j=1}^{k} |t_i \cap C_j|, \text{ with } C_j \in vclusters(D_m)$$

$$correlation(t_i, C_j) = p(t_i|C_j) = \frac{|t_i \cap C_j|}{\sum_{k=1}^{n} |t_k \cap C_j|}$$

$$freq(t_i, D_m) = \frac{occurrences(t_i, D_m)}{\sum_{i=1}^{n} occurrences(t_i, D_m)}$$

We follow the next strategy:

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters C_1, C_2, C_3
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    $score(t_i) = w_d * \sum_{j=1}^{3}(correlation(t_i, C_j) * freq(t_i))$
  end for

end for
recommend tags with the highest score

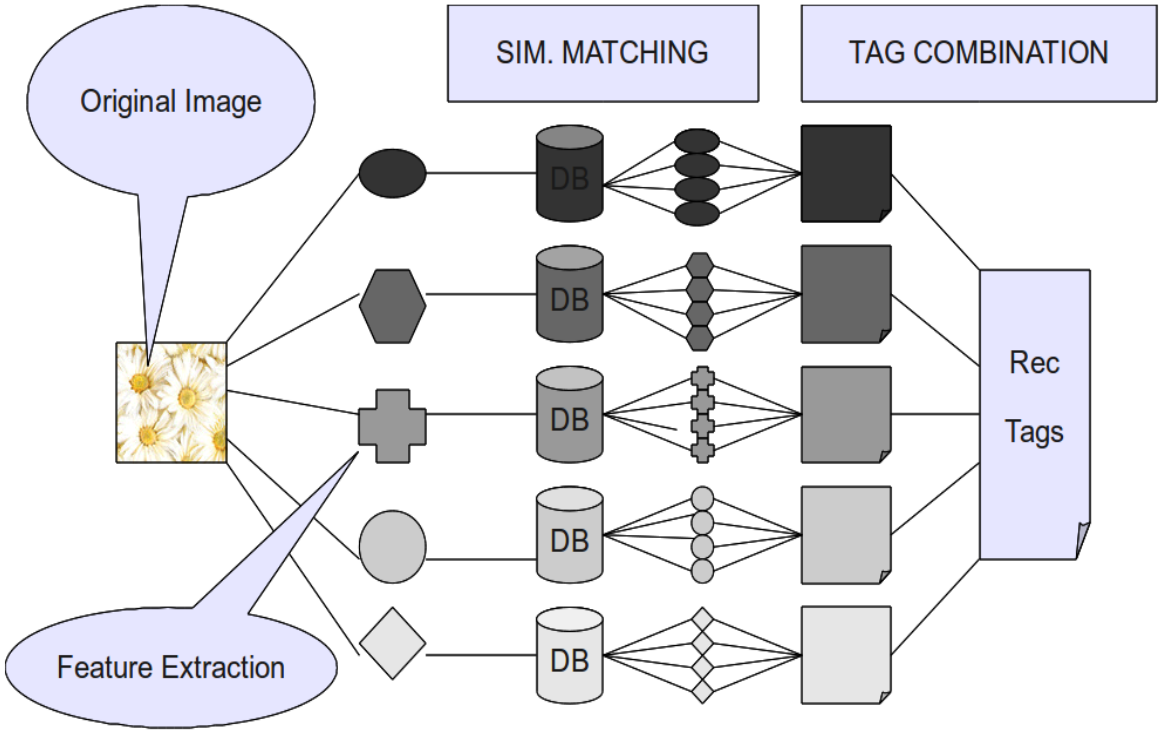


Figure 4.3: Similarity search with different descriptors

The figure 4.3 shows how the similarity search is performed. We can see that a different search is performed for each descriptor. Given an image to analyze, the first step is the extraction of the MPEG-7 features. Then, for each feature, we look for the nearest clusters in the dataset. As we explained before, only the centroids are used to perform the similarity search. For each of the clusters found, we obtain the tags that appear in at least one image belonging to the cluster. At this point we compute the score for each of the tags found using their global frequency and their correlation with the clusters in which they appear as explained before. In the figure, this step is called Tag Combination. The recommended tags are simply the ones with the highest score.

We show the results obtained with this strategy running the application for DAT1 and DAT2:

	Scalable Color Descriptor		*Combination*	
	Class	Tag Acc.	Class	Tag Acc.
Dataset 1	0.284	0.271	0.305	0.309
Dataset 2	0.08	0.091	0.052	0.116

We can see that the combination of different MPEG-7 descriptors outperforms the use of a single descriptor. The results obtained with this technique are not accurate enough, mostly for the second dataset where the sparsity of tags is higher. It is due to the fact that the visual clusters obtained aren't always semantically related as we can see in figure 4.4 and 4.5. Therefore tags and clusters usually have a low correlation and the most popular tags in the dataset tend to be recommended. We need to obtain more semantically related clusters in order to improve the efficacy of the recommendation.

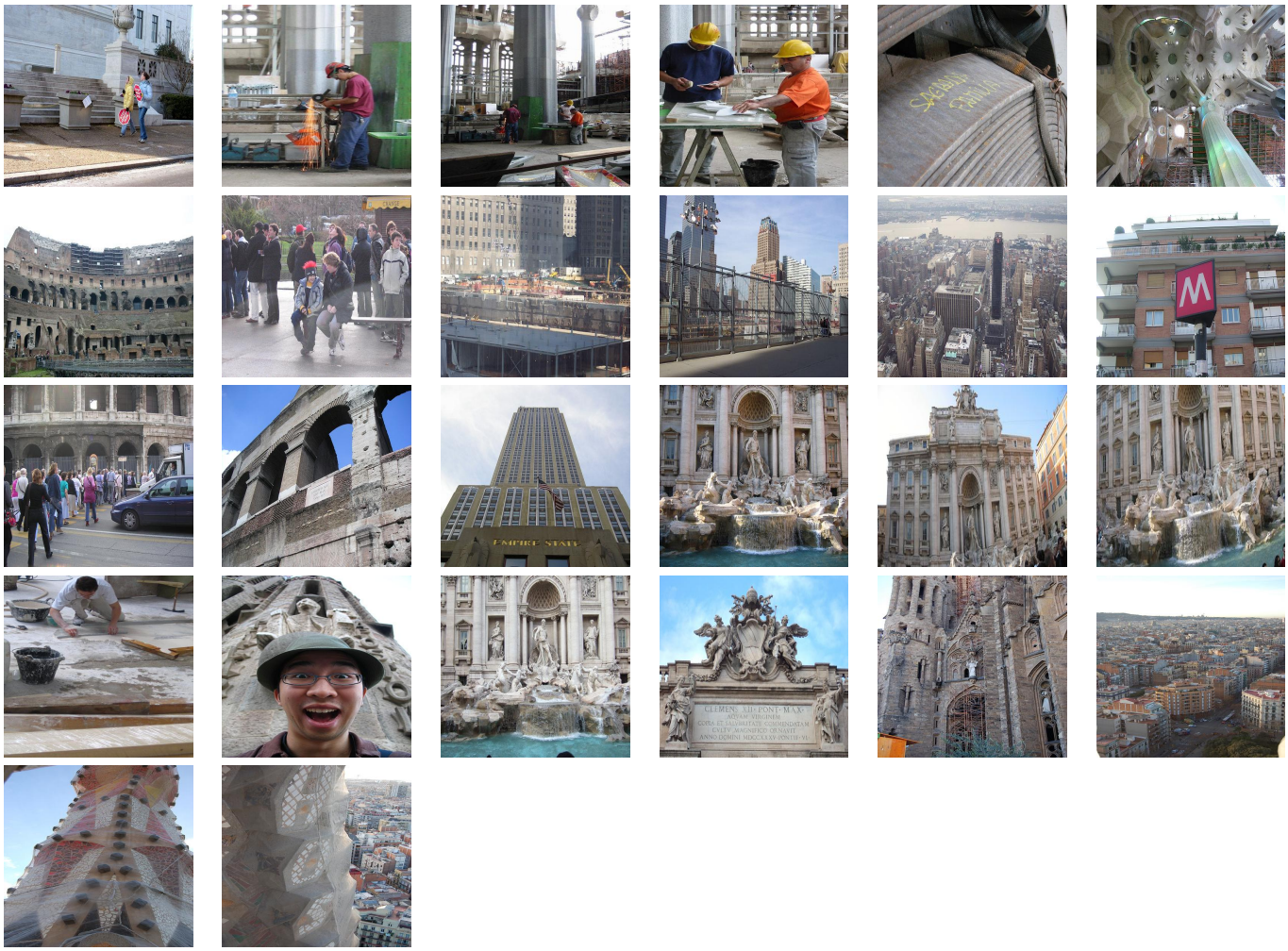


Figure 4.4: Example of a Cluster of DAT1 obtained with the Color Structure Descriptor

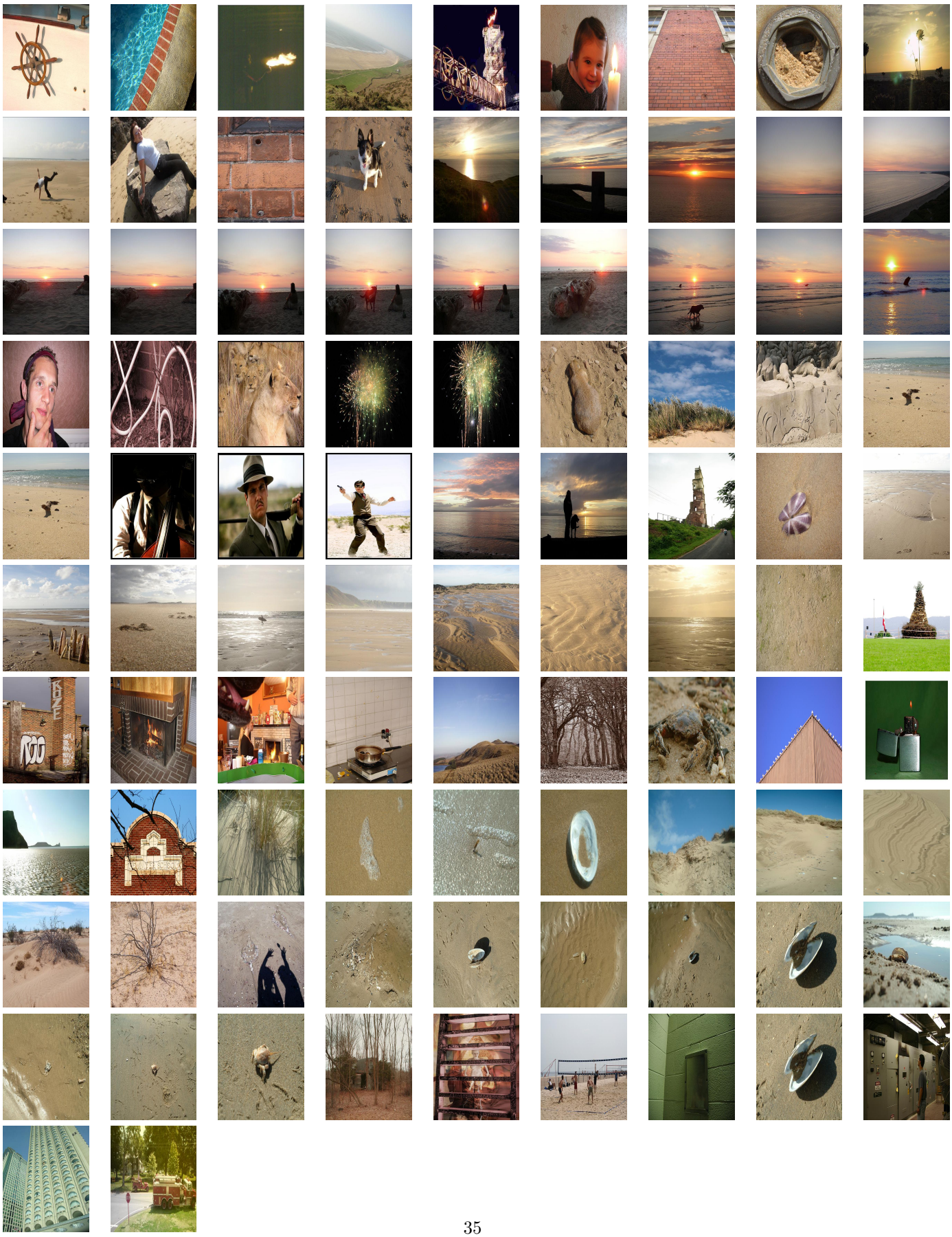


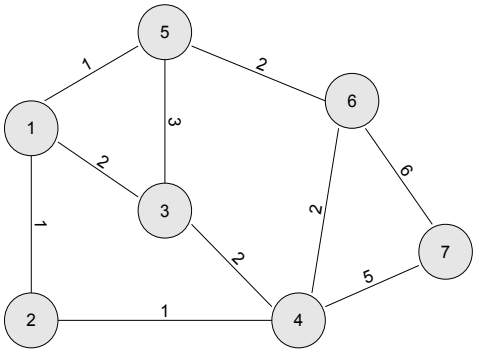
Figure 4.5: Example of a Cluster of DAT2 obtained with the Color Structure Descriptor

4.5 Textual + Visual Clustering

As we have seen in the previous experiment, working with visual clusters obtained directly with MPEG-7 features is not enough for our pourpose. It is due to the fact that MPEG-7 descriptors may be close for images tagged in absolutely different ways. As we can see in the visual cluster showed in figure 4.4, images tagged with 'colosseo', with 'fontanaditrevi' and with 'sagradafamilia' appear together in the same cluster. Also, in the visual cluster showed in figure 4.5, images containing the tag 'brick' appear together with images showing a sunset. In fact, those images are visually similar, they contain really similar colors. Therefore, a visual clustering technique does not allow us to find tags strongly related to clusters. In order to avoid this problem, we regroup the images tagged in a similar way. The idea is to obtain the visual clusters in two steps. In the first step, we perform a textual clustering of the images forming the training set. In the second step, we proceed with the visual clustering based in the MPEG-7 descriptors. We want to regroup images visually similar but also with a similar semantic meaning.

4.5.1 Textual Clustering

In this section, we first make a clustering based on the textual information of the image. We use the software k-Metis in order to partition the images. This software suits us because it is scalable and therefore we are able to use it when trying to implement a larger scale application. In order to use this software we have to create a graph-like input.
We show here an example proposed in ??? by George Karypis and Vipin Kumar in order to understand the format needed for the input of k-metis. In our case, the nodes represent the images while the arcs represent the Semantic Similarities explained later.



INPUT FILE:
7 11 1
5 1 3 2 2 1
1 1 3 2 4 1
5 3 4 2 2 2 1 2
2 1 3 2 6 2 7 5
1 1 3 3 6 2
5 2 4 2 7 6
6 6 4 5

Figure 4.6: Example of input for k-metis

The numbers in the first line of the input file correspond to the number of nodes, the number of edges and the kind of graph. In this case, it is a weighted graph weighted on edges. The connections of each node are described in a different line of the input file. For example, the second line describes the connections of the node 1: it is connected to node 5 with weight 1, to node 3 with weight 2 and to node 2 with weight 1.

In our case, we consider that the images are the nodes of the graph and that they are not weighted. On the other hand, the edges are weighted, and the weights are the semantic similarities between images. This semantic similarity represents how close the tags of two images are. Counting the commom tags of two images happened to be a weak measure, in fact tags like colosseo, coliseum, coliseo would be considered different. Other techniques like stemming are not really suitable in this case because most of the time the problem remains unsolved. We need a way to measure the similarity between words or parts of the words. We decide to adopt a technique based on the article ??????

Semantic Similarity

Let T_A and T_B be the tags of the images A and B repectively:

$$Tags_A = \{t_{A1}, ..., t_{Am}\}$$
$$Tags_B = \{t_{B1}, ..., t_{Bm'}\}$$

An n-gram is a subsequence of n items from a given sequence. Trigrams are a special case of n-gram where n=3. In this case we consider the sequence to be a tag and the items to be letters. For each tag we obtain a set of trigrams which correspond to subsequences of three letters.
For example, given the tag 'italian' we obtain the set of trigrams { 'ita', 'tal', 'ali', 'lia', 'ian' }
We extract the trigrams for T_A and T_B, we obtain the two sets:

$$Trigrams_A = \{tr_{A1_1}...tr_{A1_i}, ..., tr_{Am_1}, ..., tr_{Am_{i'}}\}$$
$$Trigrams_B = \{tr_{B1_1}...tr_{B1_j}, ..., tr_{Bm'_1}, ..., tr_{Bm'_{j'}}\}$$

We compute the semantic similarity as follows:

$$Sim(A, B) = \frac{2*(Trigrams_A \cap Trigrams_B)}{|Trigrams_A| + |Trigrams_B|} \tag{4.2}$$

We now show an example of the computation of this measure:

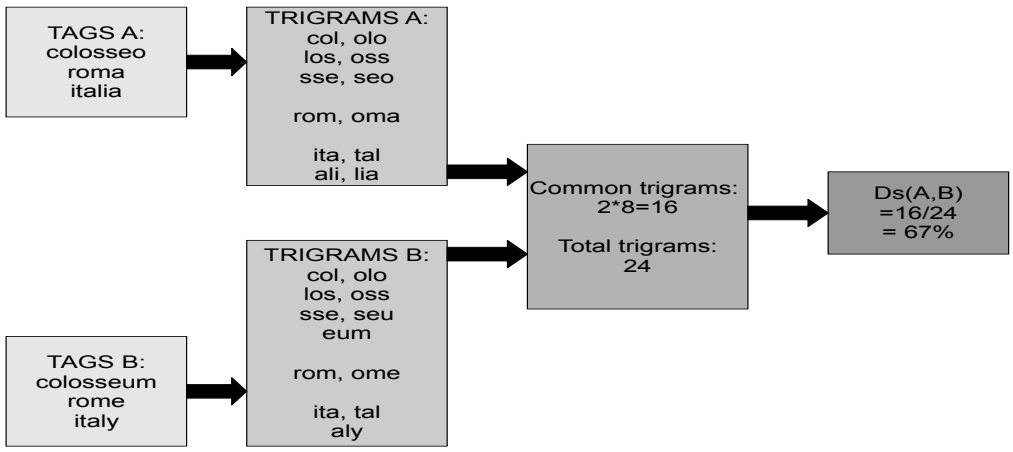


Figure 4.7: Example of semantic similarity

In this example we can see the procedure followed to compute the semantic similarity between two images A and B. It is important to note that if we had considered only the common tags as similarity measure, these two images would have similarity equal to zero. This technique also leads to to consider false similarities, for example the two tags 'roma' and 'romania' would be considered 57% similar. Nevertheless this method has shown a better performance than counting the common tags, i.e. the clusters obtained are more homogeneous.

The graphs obtained are dense an the weights are numbers between 0 and 1. The k-metis software needs the weights to be integers, so we proceed as follows: we multiply the weights * 1000 to keep a high precision, if still a weight is smaller than 0 we ignore it and consider the similarity as 0.

For DAT1, we obtain a graph with 1333 nodes and 356976 edges while for DAT2 we have 1000 nodes and 261395 edges. Once we have represented the sets of images as graphs, we are ready to use the software. We need to determine the parameter k which is the number of clusters we want to obtain. We run k-metis with k=25, 50, 75. Therefore we have clusters of approx. 50, 25 and 15 images respectively.
We observe that the clusters obtained with k=50 are the best ones. These clusters are specially good in the case of the DAT1. We also observe that the variance of the amount of images per cluster is lower than the one obtained with k-means. But this technique does not use centroids, which means we do not have a representative element for each cluster.

We now show two examples of textual clusters obtained with this method:

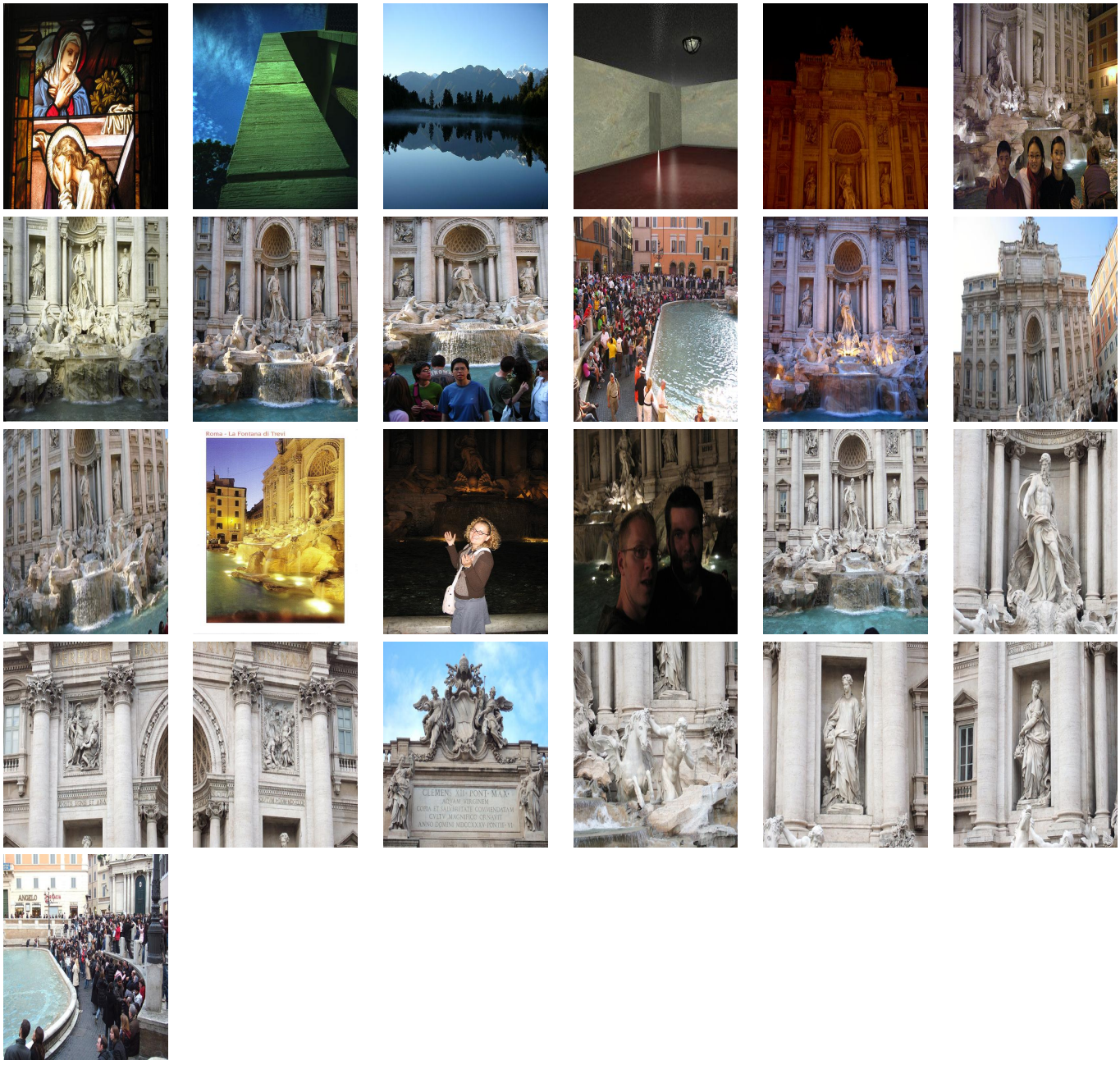


Figure 4.8: Example of a textual cluster of DAT1 obtained with k-metis

In this figure 4.8 we show a textual cluster obtained with k-metis. We can see that most of the images are related to the Trevi Fountain. It is due to the fact that thay are tagged in a similar way. This cluster helps in our pourpose of recognizing representative bulidings. Nevertheless, this cluster is not absolutely homomgeneous, in fact some of the images that have fallen into this cluster are not related at all with the Trevi Fountain.

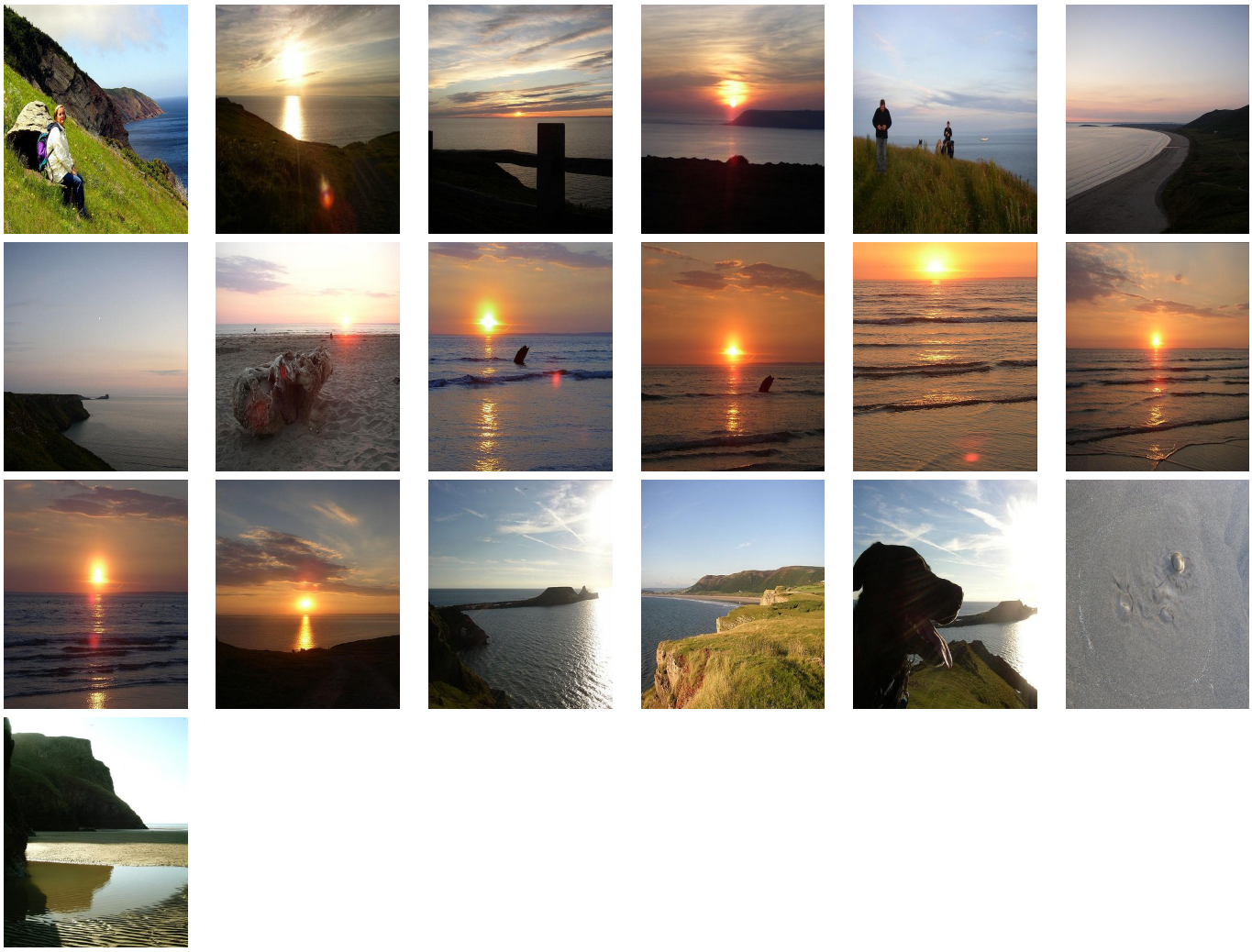


Figure 4.9: Example of a textual cluster of DAT2 obtained with kmetis

In this figure 4.9 we show a textual cluster obtained with k-metis. We can see that all the images represent similar landscapes. Once again, it is due to the fact that thay are tagged in a similar way. This cluster helps in our pourpose of recognizing panorama photos or landscapes. We can imagine that some of the tags present among these images are ‘landscape’, ‘ocean’, ‘sea’, ‘nature’ or ‘sunset’.

4.5.2 Visual Clustering

At this point, we have clusters formed mainly by images that are semantically similar. In order to establish a strong relation between visual image feature and tags, we need a visual clustering similar to the one used for the baseline. For each textual cluster and each MPEG-7 descriptor, we use a k-means clustering algorithm to obtain visual clusters. These clusters are formed by images both semantically and visually related. Each of the textual clusters has more or less 25 photos. We consider adequate to run the k-means algorithm with k=4. Hence we obtain 200 visual clusters composed of approximately 6 images. These clusters are suitable for us because we have created a strong link between tags and visual content of the image. Given an image to analayze, we can use their centroids to search for the nearests clusters in the dataset.

Recommendation Strategy

In the next experiments, we follow a recommendation strategy very similar to the one explained before. The only difference is that we compute the tag frequencies and the correlation between tags and clusters considering the new visual clusters instead of the ones obtained for the baseline:

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters C_1, C_2, C_3
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    $score(t_i) = w_d * \sum_{j=1}^{3}(correlation(t_i, C_j) * freq(t_i))$
  end for
end for
recommend tags with the highest score

We show here the results with this new clusters and the ones obtained previously for the baseline:

	Baseline			*Text. + Visual Clusters*		
	Class	Said Categ.	T.A.	Class	Said Categ.	T.A.
Dataset 1	0.305	0.582	0.309	0.301	0.627	0.335
Dataset 2	0.052	0.669	0.116	0.086	0.683	0.123

We can see a slight improvement in the Tagging Accuracy for DAT1 and for DAT2. When we look the clusters in detail, we can see that globally they are way more homogeneous that the ones found for the baseline. Nevertheless, the accuracy of the recommended tags does not increase accordingly. It is due to the fact that some of the clusters obtained are heterogeneous. This fact implies that the images forming the cluster are distant from the centroid and have few common tags. The result is that the tags recommended by these clusters are usually too specific and unrelated between them. For these clusters, there has not been established a strong relationship between tags and image features.

We show here two homogeneous clusters obtained with this combination of textual and visual clustering:

Figure 4.10: Example of a Cluster of DAT1 obtained with the Color Structure Descriptor

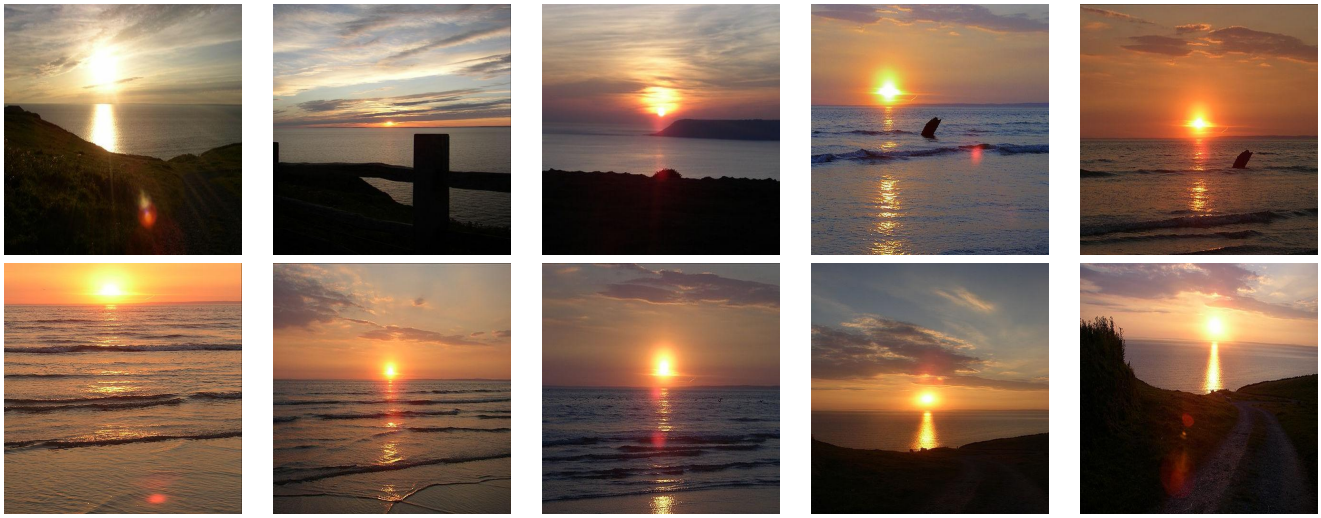


Figure 4.11: Example of a Cluster of DAT2 obtained with the Color Structure Descriptor

4.5.3 Tuning the number of visual clusters

We need to tune the different parameters that we are using in this strategy. In this section we want to fix the number of clusters m that we need to look for when we analyze a given image. The idea is that considering too few clusters causes the apparition of too specific tags. On the other hand, looking for too many clusters makes the most popular tags tend to appear and most likely introduces errors. For example, when we examine the clusters found for the Dataset 1, only three of them contain images of the Trevi Fountain. Therefore, if we want to recommend tags for an image of the Trevi Fountain, considering too many clusters probably causes a decrease of the precision of the recommended tags. We pretend to find empirically the number of clusters we need to look for, in the next experiments we run the application for both DAT1 and DAT2 with the values $m = \{3, 5, 10\}$

Recommendation Strategy

We use the same strategy than in the previous section with the values m=$\{3, 5, 10\}$:

for $d \in descriptors$ **do**
  find the m nearest visual clusters $C_1...C_m$
  for $t_i \in \bigcup_{j=1}^{m} tags(C_j, d)$ **do**
    $score(t_i) = w_d * \sum_{j=1}^{m}(correlation(t_i, C_j) * freq(t_i))$
  end for
end for
recommend tags with the highest score

We show here the results obtained for the different values of m:

	Baseline		*3 clusters*		*5 clusters*		*10 clusters*	
	Class	T.A.	Class	T.A.	Class	T.A.	Class	T.A.
Dataset 1	0.305	0.309	0.301	0.335	0.310	0.336	0.320	0.333
Dataset 2	0.052	0.116	0.086	0.123	0.076	0.118	0.050	0.122

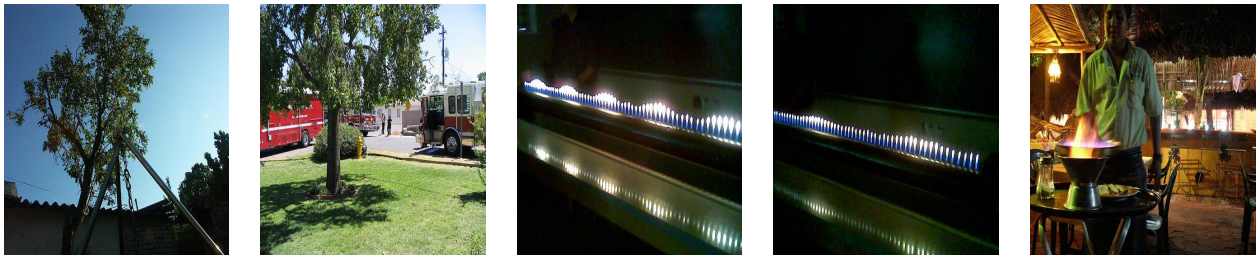
We can see that there is not a great difference in the performance for the three values of m. Therefore, from now on we fix m=3 because the recommendation strategy is faster. In fact, fixing $m = 3$ means computing less of the probabilities and correlations explained before. When we analyze the proposed tags, we can see that they tend to be too specific. In fact, some tags only occur in one cluster, these tags are probably recommended because they have a correlation of 100% but they are hardly suitable for any image. We know that looking for more visual clusters does not fix the problem so we need to find a way to propose more general tags.

We now show a graphical example of the strategy followed in order to recommend tags. We show the query image and the three visual clusters found with the Scalable Color Descriptor:

QUERY IMAGE:

CLUSTER 1:

CLUSTER 2:



CLUSTER 3:

Figure 4.12: Clusters found with the Scalable Color Descriptor

4.5.4 Freq. cluster visuale vs freq. cluster text

As we have explained in the previous section, the tags proposed at this point are too specific. In order to solve this problem we compute the correlations between clusters and tags considering the textual clusters instead of the visual clusters. As the textual clusters are bigger, tags with high correlations are not tags with low global occurrences anymore, so more general tags are proposed.

We modify the correlation formula as follows:

$$\begin{aligned} tclusters(D_m) &= \{TC_1..TC_k\} \text{ is the set of textual clusters found with kmetis} \\ tcorrelation(t_i, TC_j) &= p(t_i|TC_j) \\ &= \frac{|t_i \cap TC_j|}{\sum_{k=1}^{n} |t_k \cap TC_j|} \end{aligned} \tag{4.3}$$

It is important to note that the frequency of each tag remains unchanged.

Recommendation Strategy

We modify the recommending strategy as follows:

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters $C_1 \subset TC_1, C_2 \subset TC_2, C_3 \subset TC_3$
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    $score(t_i) = \sum_{j=1}^{3}(tcorrelation(t_i, TC_j) * freq(t_i))$
  end for
end for
recommend tags with the highest score

We now show the results obtained with this method:

	Baseline			*Freq. visual*			*Freq. text*		
	Class	Said Categ.	T.A.	Class	Said Categ.	T.A.	Class	Said Categ.	T.A.
Dataset 1	0.305	0.582	0.309	0.301	0.627	0.335	0.332	0.641	0.325
Dataset 2	0.052	0.669	0.116	0.086	0.683	0.123	0.162	0.665	0.118

We observe that the percentage of well recommended tags decreases slightly while there is an increase of the classification accuracy. Even if the global performance remains similar, the recommended tags are not as specific as in the previous section. The accuracy obtained is not satisfactory at the moment, therefore we need to find other strategies that perform better than this one.

4.5.5 Visual Cluster Filter

As we have explained before, the datasets we are working with are very noisy. We tried to avoid the consequences of this noise using clustering techniques. But we have realized that not all of the clusters behave equally. In fact some of the clusters do never recommend good tags while others have a great accuracy. This means that for some visual clusters, visual content and tags are strongly related. On the other hand, some visual clusters are heterogeneous and the images forming those clusters are barely related. The idea is that those clusters don't perform correctly. Our goal in this section is to rule out visual clusters with a low performance.

First Filter

In this firt step of the filtering we try to measure the accuracy of each visual cluster. We count how many times a visual cluster C has been used and how many good recommendations this visual cluster C has done.

Let C be a visual cluster, then

$$p(C) = \frac{\mid \text{good recommendations} \mid}{\mid \text{total recommendations} \mid} \tag{4.4}$$

We call $p(C)$ the probabilty of success for the cluster C. The higher $p(C)$ is, the higher are the chances of the visual cluster C to recommend good tags.
We use a machine learning based approach in order to train our recommeder system. We want to establish the probabilties of success for as many visual clusters as possible. We proceed to determine these values in three iterations. Depending on the reliabilty of a given visual cluster, we decide whether to keep it or not for the next iteration of the learning process.

We now explain each step of the process in detail:

- iter 0: We run the recommending system for the entire validation sets (iter=1). At the end of the first run, we have probabilities of success for most of the visual clusters. The problem is that not all the visual clusters have been used so we do not have the probabilites of success for all of the visual clusters.

- iter 1: We proceed to run the recommending system again ruling out those visual clusters that have a coefficient of success equal to zero (iter=2). In other words, we rule out those visual clusters that appeared but never made a good recommendation. At the end of this run, we hope to have a coefficient of success for those visual clusters who did not appear before.

- iter 2: Just like in the previous iteration, we rule out those visual clusters that appeared but never made a good recommendation.

At the end of the three runs, we have probabilites of success for most of the visual clusters. These probabilities have changed during these three different iterations of the learning process. We observed that the fourth iteration of the process barely changed these coefficients, so there is no use in doing more than three iterations.

In the following experiment, we use the probabilities of success obtained in the learning process just explained. We adopt a simple strategy which consists of considering only those visual cluster that have a probability of success greater than a given limit L ($p(C) > L$). In others words, we don't consider the visual clusters that have $p(C) <= L$. We study the performance of our recommender for two different values of L:

- $L = 0.0$ in this case we consider those visual clusters that have $p(C) > 0.0$
- $L = 0.1$ in the second case we consider those visual clusters that have $p(C) > 0.1$

Recommendation Strategy

We modify the recommending strategy as follows:

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters C_1 with $p(C_1) > L$, C_2 with $p(C_2) > L$, C_3 with $p(C_1) > L$
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    $score(t_i) = w_d * \sum_{j=1}^{3}$ occurrences of t_i in C_j
  end for
end for
recommend tags with the highest score

	Baseline			$L \geq 0$			$L \geq 0.1$		
	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.
Dataset 1	0.305	0.582	0.309	0.342	0.67	0.346	0.338	0.731	0.405
Dataset 2	0.052	0.669	0.116	0.237	0.725	0.132	0.391	0.750	0.226

We can see a significative increase of the performance. It is important to see that reducing the number of visual clusters we work with increases both the accuracy and the efficiency of the recommendations. Nevertheless this increase might be dataset dependent, because some of clusters we are ruling out might be homogeneous visual clusters and still do not fit well with the images in the validation set.

Another thing we need to consider is the accuarcy of the probabilities obtained before. Those clusters who have often been used are likely to have a coefficient in accordance with their performance. Instead, the coefficent given to visual clusters who have been barely used may not be reliable.
Therefore in the next experiment, we consider the visual clusters with a low number of occurrences even if their p(C) is also low. We compare the three following cases:

- case 1: we consider only the visual clusters with p(C) > 0.1 (this case corresponds to the previous experiment)
- case 2: we consider the visual clusters with p(C) > 0.1 or occurrences < 5
- case 3: we consider the visual clusters with p(C) > 0.1 or occurrences < 10

	Baseline			*case 1*			*case 2*			*case 3*		
	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.
Dat 1	0.305	0.582	0.309	0.338	0.731	0.405	0.337	0.725	0.400	0.335	0.701	0.390
Dat 2	0.052	0.669	0.116	0.391	0.750	0.226	0.381	0.742	0.217	0.381	0.730	0.193

We conclude it is better to ignore the visual clusters with a low number of occurrences. We might be losing some information but as we can see in this experiment, ruling out these clusters increases the performance of the tag recommender.

Second Filter

At this point, we have a coefficient of accuracy for each visual cluster. But our intuition is that the accuracy of a visual cluster depends on the tag that it is recommending. Therefore we consider more appropriate to measure, given a visual cluster C, the probability of each individual tag t to be well recommended. Let $p(t|C)$ be that probability.

For the next learning process, we work only with the visual clusters with a probability of success greater than zero ($p(C) > 0$). Even if the recommender worked better when we kept only the visual clusters having $p(C) > 0.1$, we consider it is better to keep all the visual clusters that have done at least one good recommendation. The reason to do so is that even the visual clusters with a low $p(C)$ could be a strongly related to some tags.

Once again, we run the recommending system for both training sets. We count the number of times that a visual cluster recommends a tag and the number of times that the recommended tag is correct. The cases where $p(t|C)$ are high correspond to the cases where there is a strong relation between image feature and tag. This was one of the objectives of the use of visual clusters and now we have a way to quantify this relation. For each visual cluster C considered, we now have a value $p(t|C)$ for each of the tags t that the visual cluster recommends. In other words, for every tag t appearing in at least one image belonging to the visual cluster C, we have a value $p(t|C)$.

In the next experiment, we use the probabilities $p(t|C)$ that we have just obtained. We adopt a simple strategy which consists of recommending tag only if its coefficient of success is higher than a given limit l. We compare the four following cases:

- case 1: we consider the tags having $p(t|C) > 0.0$
- case 2: we consider the tags having $p(t|C) > 0.1$
- case 3: we consider the tags having $p(t|C) > 0.2$
- case 4: we consider the tags having $p(t|C) > 0.3$

Recommendation Strategy

We modify the strategy as follows, we run the recommender for the values l={ 0.0, 0.1, 0.2, 0.3 }:

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters C_j with $p(C_1) > 0$
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    if $p(t_i|C_j) >$ L **then**
      $score(t_i) = w_d * \sum_{j=1}^{3}$ occurrences of t_i in C_j
    end if
  end for
end for
recommend tags with the highest score

It is important to note that given a visual cluster C, we use $p(t|C)$ in order to filter the tags t proposed by C but not to compute the score of t. In fact, including $p(t|C)$ in the score formula didn't increase the accuracy of our recommender.

	Baseline			*case 1*			*case 2*			*case 3*			*case 4*		
	Class	S. C.	T.A.	Class	S. C.	T.A.	Class	S. C.	T.A.	Class	S. C.	T.A.	Class	S. C.	T.A.
Dat 1	0.305	0.582	0.309	0.338	0.731	0.405	0.347	0.809	0.444	0.390	0.845	0.459	0.483	0.802	0.406
Dat 2	0.052	0.669	0.116	0.391	0.750	0.226	0.439	0.754	0.236	0.491	0.701	0.182	0.473	0.581	0.129

We can see a global improvement in the recommending performance and in the classification. The tagging accuracy obtained for Dataset 1 is more than satisfactory, in fact almost half of the original tags are proposed. In the case of the second dataset, we recommend a 24% of the original tags in the case where we fix $p(t|C) > 0.1$. Considering the variety and sparsity of the tags with which the images are labeled, it is a good performance. For the next experiments, we fix the value $p(t|C) > 0.1$ because it is the value that has showed a better overall performance.

4.5.6 Examples

Examples of good recommendations from dataset 2

ORIGINAL TAGS:
auvergne
cantal
aphanasis
disparition
abandon
margeride
lozere
woods

RECOMMENDED TAGS:
woods
fire
hiking
autumn
snow
brick
water
square
leaves
fall

ORIGINAL TAGS:
brick
truck
paint

RECOMMENDED TAGS:
brick
red
fire
wall
building
newyorkcity
nyc
brooklyn
portrait
sign

Even though few of the original tags are proposed in these two examples, we can see a good performance of our recommender. In the case of the first image, it is not surprising because the training set contained many similar images. Therefore the recommendation was likely to be accurate.

Examples of bad recommendations from dataset 2

ORIGINAL TAGS:
fire
furnace
burn
flames

RECOMMENDED TAGS:
woods
fire
winter
snow
beach
sea
trees
bluesky
forest
brick

In this case, the recommender performs poorly, nevertheless the image wasn't easy to tag. We can see that when the recommendation is not accurate, the most popular tags tend to appear.

ORIGINAL TAGS:
church
trinidad
cuba
bluesky

RECOMMENDED TAGS:
bluesky
fire
blue
yellow
sea
red
sky
nature

The recommender does not recommend a good set of tags. It is due to the fact that there are few images of similar buildings in Dataset2, we expect this kind of image to be well recommended when we deal with a bigger dataset. The combination of MPEG-7 and SURF features should also improve the accuracy of the recommended tags in these cases.

ORIGINAL TAGS:
bluesky
sky
spring
magnolia
blossoms

RECOMMENDED TAGS:
bluesky
brick
sea
fire
yellow
summer
london
red
clouds
blue

This is another case of poor performance even though some good tags are proposed. We can see that 'brick', 'sea', 'fire' or 'london' should not have been proposed. In the case of the tag 'clouds' it is easy to see that these flowers have been confused with clouds because of their similar color, form and context.

Special cases

OR. TAGS:
nature
hiking
nationalparks
mountrainier
seattle
seattlevacation
woods
july2006

REC. TAGS:
bluesky
brick
yellow
red
sky

REC. TAGS:
bluesky
fire
sunset
sky
blue
summer
clouds
nature
woods
burn

In this case these two images have the same original tags even though the images are different. In the first case the recommender performs poorly while mostly good tags are proposed in the second case. We cn see how the woods are confused with a fire because of the similar forms.

OR. TAGS:
vertical
lookup
tower
window
bricks
mortar
sky
bluesky
summer
louisebarr

REC. TAGS:
sunset
sand
beach
brick
sea
summer
fire
desert
birthday
sky

REC. TAGS:
beach
sand
brick
sea
fire
desert
arizona
building
wall

In this case these two images also have the same original tags. These are difficult images to tag because of the angle with wich the photos have been taken. In both cases it seems that this church has been confused with a beach panorama because of the colors and the edges.

Chapter 5

Visual Clusters + SURF Features

Including SURF features in a real time recommending system is a challenging problem. In fact, the different similarity measures exposed in section 3 have a great computational cost. In other works, clustering techniques are adopted in order to reduce the amount SURF keypoints, but in most of the cases the datasets used in their experiments are small. We now show the computational costs of the different metrics exposed in section 3.

5.1 COST ANALYSIS

Let C_m be the cost of a SURF vector distance computation:

$$C_m = \theta(1)$$

Let N be the total number of images of the trainng set:

N_1=1333 for DAT1
N_2=1000 for DAT2

Let k be the number of SURF keypoints per image extracted with the software by Herbert Bay, Tinne Tuytelaars and Luc Van Gool:
k_A is the amount of keypoints extracted for image we want to analyze
k_1=486532/1333=364.99 keypoints extracted per image for DAT1
k_2=486532/1000=364.99 keypoints extracted per image for DAT2
k_{t1}=75135/1333=56.37 keypoints extracted per image with a threshold t=300000 for DAT1
k_{t2}=75135/1333=56.37 keypoints extracted per image with a threshold t=300000 for DAT2

We work with the keypoints extracted with threshold t=300000 because even if we are loosing some information, the set of keypoints is a lot smaller.

Cost of the search of the image with the nearest match in DAT1:

$\theta(D_1) = N * k_A * k_{t1} * C_m$

Cost of the search of the image with the n nearest matches in DAT1:

$\theta(D_n) = N * k_A * k_{t1} * C_m$

Cost of the search of the image with most matches in DAT1:

$\theta(matches(A, DAT1)) = N * k_A * k_{t1} * C_m$

Cost of the search of the image with most bimatches in DAT1:

$\theta(bimatches(A, DAT1)) = N * (k_A * k_{t1} + k_{t1} * k_A) * C_m = N * 2(k_A * k_{t1}) * C_m = 2 * N * k_A * k_{t1} * C_m$

These techniques are obviously not scalable. For example, a linear search of the nearest match in Dataset1 composed of 1333 images takes a few minutes. The lenght of these searchs is not acceptable for our project so we must find a way to include the SURF features and still recommend tags in a reasonable time even if it means to sacrifice some efficacy or accuracy. We propose in this section a technique that combines the use of MPEG-7 features and SURF keypoints.

5.2 Candidate Selection

In our first experiments, we realized that the amount of bimatches between two images is the stronger matching technique. The problem is that it is also the most expensive. Hence, given an image to analyze, we try to reduce the number of images to which the image must be compared with this technique. The idea is to compute the SURF bimatches only to a reduced set of images, the most promising candidates.

As explained in section 3, the smaller the distance between two keypoints that match is, the higher are the chances of this match to be accurate. Therefore, finding the nearest matches in the dataset is a good technique to choose promising images. Still the amount of SURF keypoints to compare is huge. The idea is that many of these keypoints are useless because they don't match with any other keypoint or because they are not related to any tag.

Filtering Strategy

Let A be an image of the training set, then

$$keys_A = \{\text{set of keypoints } k \text{ extracted from A}\} \tag{5.1}$$

The idea of this first filtering is to compare every SURF keypoint in the dataset with all the others. We keep a keypoint k_1 when it matches with any other keypoint k_2 in the dataset and the euclidean distance between their descriptor vector is less that a given limit T.

We proceed to filter these keypoints as follows:

for each image $A \in$ Training Set **do**
  for $k \in keys_A$ **do**
    compare to all the other keypoints in the dataset
    if a match is produced with distance $< T$ **then**
      keep k
    else
      delete k
    end if

```
    end for
  end for
```

With this filter we have reduced significatively the amount of SURF keypoints in both datasets:

DAT1: from 75135 to 4310
DAT2: from 48163 to 1554

In the next step of the filtering, we only keep a keypoint if it matches with another keypoint (previous filter) and if the two compared images share at least one tag. Once again, we pretend to establish a relation between tags and image features:

```
for each image A ∈ Training Set do
  for k ∈ k_A do
    compare to all the other keypoints in the dataset
    if a match is produced with an image B with distance < T then
      if images A and B share at least one tag then
        keep k
      end if
    else
      delete k
    end if
  end for
end for
```

With this second filter we have reduced even more the amount of SURF keypoints in both datasets:

DAT1: from 4310 to 3013
DAT2: from 1554 to 1196

At this point, a linear search of the nearest n matches in the dataset only takes a few seconds, which is an acceptable time for us. We work with the distances D_1, D_3 and D_5 explained in chapter 3 in order to find the most promising candidates.

5.2.1 Candidate Selection Strategy 1

We propose the following procedure in order to find candidates. Given an image A to analyze, we look for the nearest m images in the training set. In this case we use the metrics D_n explained in chapter 3 in order to compare each keypoint extracted from A with all the keypoints in the filtered keypoint dataset. Once we have those m images, we compute the amount of bimatches between the image A and each of the m images found.

```
search in the filtered keypoint dataset for the m images with the smallest D_n
for each image B of the m images do
  compute bimatches(A, B)
end for
order the m images by the amount of bimatches found
```

We show here an example of this procedure with m=5 and working with the distance D_1.

number of bimatches		10	3	3	2	1
Query image				*nearest matches*		

Figure 5.1: Similarity search based on the distance D1

This technique allows us to search for keypoints in an greatly reduced dataset. It is important to understand that some of the keypoints we have rejected might have been useful. In fact, as explained in section 3, the distance between two matches is not always a good criterion. Some bimatches may have really different feature vectors and still be accurate. With this filter, we are rejecting all the information that this matches could provide us. Therefore we obtain a better computational cost at expense of efficacy. We are also aware of the fact that this filtering method is hardly feasible for a larger dataset. As one of our long term goals is to extend these techniques to larger datasets, we need other ways to find candidates.

5.2.2 Candidate Selection Strategy 2: Looking in the Visual Clusters

We now propose a different filtering of the SURF keypoints independent from the one just explained. During the experiments of section 4.5, we realized that the clusters obtained with the search based on MPEG-7 descriptors are also a good way to choose candidates. In fact, those clusters usually contain images who would likely produce a great number of SURF bimatches. Our hope is that finding only one cluster containing exact matches should be enough to identify an image via SURF keypoints.

With the similarity search based on MPEG-7 descriptors, we obtain three clusters for each descriptor containing each 6 images approximately, therefore we obtain around 6*3*6=108 images. This amount of images is still too big. If we were to compute the SURF bimatches for each of these images, the computational time would be too high for us. We still need to reduce the set of candidates. Therefore we propose a new filtering similar to the one explained in the previous section. In this case we compare the keypoints extracted from the images of a same visual cluster between them. We only keep those keypoints that matched with any other keypoint of the same visual cluster.

Filtering Strategy

We proceed to filter these keypoints as follows:

```
for d ∈ Descriptors do
  for vc ∈ Visual Clusters do
    for each image of the visual cluster do
      for each keypoint of the image do
        if matches with another keypoint extracted from an image of the visual cluster then
          keep the keypoint
        else
          delete the keypoint
        end if
      end for
    end for
  end for
end for
```

Once this filter is done, we have a reduced set of keypoints for each visual cluster (32 keypoints approx. per visual cluster). These keypoints are representative of each visual cluster and strongly related to tags because of the fact that the visual clusters are obtained from bigger textual clusters. In this case the distance between two keypoints is not considered in order to decide whether to reject them or not as we did in the previous filter. This means that the SURF keypoints kept with these two filters may be different between them. Hence a combination of both techniques should increase the overall performance of the recommending system.

Candidate Selection Strategy 2

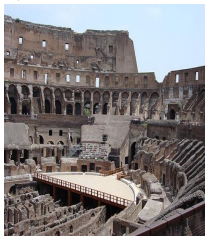
Given an image A and an MPEG-7 descriptor, we look for the three nearest visual clusters in the dataset. For each of the clusters found, we select the m images with the smallest distance D_n with the image A. We then compute the amount of bimatches between the image A and each of these three selected images. Therefore we compute the amount of bimatches between the image A and 6*3*m candidates (6 MPEG-7 descriptors, 3 clusters per descriptor and m images per cluster).

We proceed as follows in order to find the nearest images with SURF features:

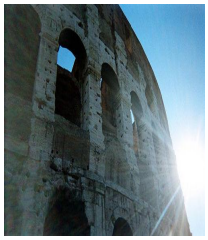
for $d \in$ MPEG-7 Descriptors **do**
  find the 3 nearest clusters
  for each cluster found **do**
    search in the filtered dataset for the m images with the smallest D_n
    for each image B of the m images **do**
      compute $bimatches(A, B)$
    end for
  end for
end for
order the $6 * 3 * m$ images by the amount of bimatches found

We observe that fixing $m = 3$ leads to the best results. The candidate selection and the computation of the amount of bimatches for each candidate takes around 10 seconds which is acceptable. Nevertheless an improvement of the efficiency of this process is needed.

QUERY IMAGE:

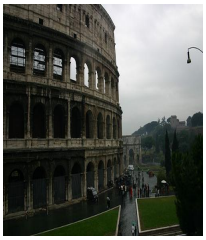


CLUSTER 1:

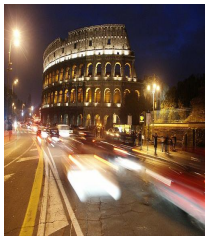


-

-



2 bimatches

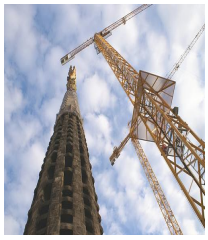


2 bimatches

10 bimatches

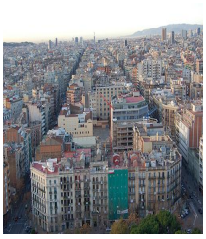
-

CLUSTER 2:

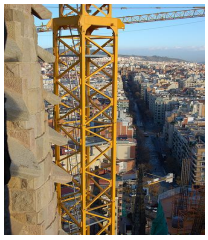


1 bimatch

-



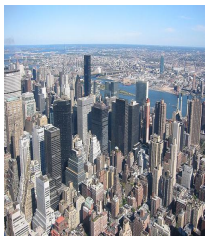
-



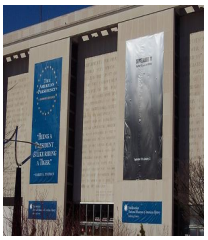
2 bimatches

1 bimatch

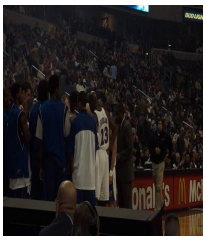
CLUSTER 3:



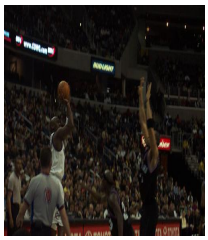
-



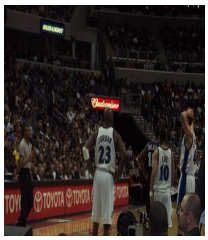
-



2 bimatches

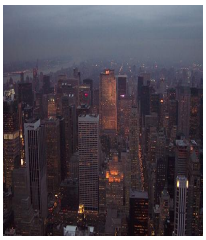


2 bimatches



-

-



-

Figure 5.2: Clusters found with the Edge Histogram Descriptor

We show here an example of the use of this technique, we can see the three clusters obtained with the similarity matching based on the Edge Histogram Descriptor. We also show the bimatches computed for the $m = 3$ best candidates of each cluster obtained with the distance D_1.

5.3 Study of the relation between SURF Bimatches and Exact Matching

Given an image to analyze, we are now able to find good candidate images and compute the amount of bimatches for these candidates. We have now to determine the amount of bimatches necessair to tell whether or not an image found is an exact match for the analyzed image. From now on, we consider that an exact match for a given image is another image that shows the exact same place or building. In order to achieve this pourpose, we work with a reduced dataset (DAT3) which is a subset of DAT1 composed of 25 manually chosen images. We have chosen these images because they are representative. These images are tagged with only one of the following tags:

- 'colosseo': 6 images
- 'empirestatebuilding': 6 images
- 'toureiffel': 6 images
- 'sagradafamilia': 5 images
- 'trevifountain': 2 images

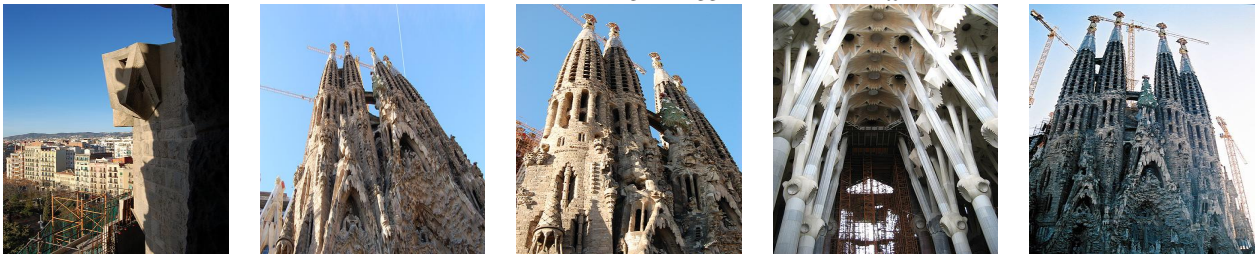
Our goal to classify and tag correctly all of the images in this reduced dataset.

We now show the 25 images forming DAT3:

Images tagged with 'colosseo'

Images tagged with 'empirestatebuilding'

Images tagged with 'toureiffel'



Images tagged with 'sagradafamilia'

Images tagged with 'trevifountain'

Figure 5.3: Images in DAT3

Procedure

For each of the 25 images in the dataset, we obtain the candidates with the two methods explained before. We decide whether or not the images returned represent an exact match.

for each image $A \in$ DAT3 **do**

find the 5 best candidates with Strategy 1
for each image $B \in candidates1$ **do**
compute $bimatches(A, B)$
decide if the image is an exact match
end for

find the 18 nearest visual clusters with MPEG-7 descriptors
for each $vc \in visualclusters$ **do**
find the 3 best candidates with Strategy 2
for each image $B \in candidates2$ **do**
compute $Bimatches(A, B)$
decide if the image is an exact match
end for
compute $\sum_{i=1}^{3} Bimatches(A, B)$ for $B \in candidates2$
decide if the cluster contains an exact match
end for

end for

For each analyzed image, we have 5 candidates corresponding to the candidates found with the first strategy and 6*3*3=54 candidates found with the second strategy. Due to the fact that the visual clusters contain similar images at a semantic and visual level, it is also interesting to compute the sum of bimatches found for one cluster. We propose the 3 following metrics:

Bim_1 : Bimatches per image obtained with strategy 1
Bim_2 : Bimatches per image obtained with strategy 2
Bim_{vc} : Sum of the bimatches found for the 3 best candidates of a given visual cluster

We now show the results obtained with each of these three methods:

Bim_1	Exact Matches
3	0.32 (9/28)
4	0.75 (6/8)
5	0.60 (3/5)
6	0.88 (7/8)
7	1.0 (6/6)
8	1.0 (9/9)
9	1.0 (1/1)
10	1.0 (3/3)
12	1.0 (1/1)
15	1.0 (1/1)
34	1.0 (1/1)

Bim_2	Exact Matches
3	0.26 (24/94)
4	0.48 (19/40)
5	0.60 (12/20)
6	0.79 (11/14)
7	1.0 (4/4)
8	1.0 (14/14)
9	1.0 (1/1)
10	1.0 (3/3)
11	1.0 (1/1)
12	1.0 (1/1)
15	1.0 (4/4)
34	1.0 (1/1)

Bim_{vc}	Exact Matches
7	0.375 (6/16)
8	0.33 (3/9)
9	0.43 (3/7)
10	0.50 (1/2)
11	0.67 (4/6)
12	1.0 (4/4)
13	1.0 (2/2)
14	1.0 (4/4)
19	1.0 (2/2)
20	1.0 (1/1)
21	1.0 (1/1)
22	1.0 (2/2)
28	1.0 (2/2)
30	1.0 (1/1)
33	1.0 (1/1)
35	1.0 (1/1)

Figure 5.4: Total Results

These three tables show the relation between the amount of bimatches and the percentage of exact matches. Now we only show the percentage of exact matches obtained for the best candidate found with each of the three methods. In other words, we obtain the candidates with Strategy1 and we only keep the image with most bimatches among these candidates (Best Bim_1). We then obtain the candidates with Strategy2 and once again we only keep the image with most bimatches among these other candidates (Best Bim_2). Finally we only keep the visual cluster with the highest sum of the bimatches (Best Bim_{vc}). We decide whether or not the images returned represent an exact match.

Best Bim_1	Exact Matches
3	0.20 (1/5)
4	0.75 (3/4)
5	0.00 (0/2)
6	0.67 (2/3)
7	1.0 (3/3)
8	1.0 (6/6)
9	1.0 (1/1)
10	1.0 (1/1)
12	1.0 (1/1)
15	1.0 (1/1)
34	1.0 (1/1)

Best Bim_2	Exact Matches
3	0.44 (4/9)
4	1.0 (5/5)
5	0.50 (4/8)
6	0.75 (3/4)
7	1.0 (1/1)
8	1.0 (11/11)
9	1.0 (1/1)
10	1.0 (1/1)
11	1.0 (1/1)
12	1.0 (1/1)
15	1.0 (4/4)
34	1.0 (1/1)

Best Bim_{vc}	Exact Matches
7	0.375 (1/2)
9	0.50 (1/2)
10	1.0 (1/1)
11	0.67 (2/3)
12	1.0 (3/3)
13	1.0 (2/2)
14	1.0 (2/2)
19	1.0 (2/2)
20	1.0 (1/1)
21	1.0 (1/1)
22	1.0 (2/2)
33	1.0 (1/1)
35	1.0 (1/1)

Figure 5.5: Best Results

As we can see, it is hard to fix a threshold that tells us whether or not there has been an exact match. In fact, there is not a clear relation between amount of bimatches and exact SURF matches. When the number of bimatches found is greater or equal than 7, we can almost be sure we found an exact match. But when we find less bimatches, we cannot be sure. Therfore, fixing a threshold too low causes many false matches while fixing it too high causes a loss of information. In the next experiments we try to tune this threshold.

5.4 Comparing D_1, D_3 and D_5

In this experiment with SURF keypoints, we compare the efficacy of the D_1, D_3 and D_5 distances. In order to evaluate this difference, we only consider the five candidates obtained with the first strategy (Strategy1).

We proceed as follows:

with D1: If an exact match is found among the five candidates found with D_1, we propose its tags as recommended tags.
with D3: If an exact match is found among the five candidates found with D_3, we propose its tags as recommended tags.
with D5: If an exact match is found among the five candidates found with D_5, we propose its tags as recommended tags.

In all of the three previous cases, we consider that an exact match takes place when 4 or more bimatches have been found for a candidate.

We evaluate the tag recommendation for Dataset1, Dataset2 and Dataset3.

	D_1			D_3			D_5		
	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.
Dataset 1	0.387	0.749	0.411	0.350	0.726	0.396	0.346	0.710	0.393
Dataset 2	0.353	0.719	0.213	0.363	0.697	0.210	0.373	0.732	0.222
Dataset 3	0.639	0.792	0.639	0.160	0.637	0.116	0.156	0.647	0.118

We can see that there is not a great difference between these 3 distances. Nevertheless, the slight better performance of D_1 means that considering only the distance of the nearest match (D_1) leads us to discover the biggest amount of exact matches. Using the D_1 distance also suits us because the search of the nearest match could be done in a more efficient way in case our application was too slow. Because of these two reasons, we only use the D_1 distance from now on.

5.5 Finding the Optimal Threshold for SURF Bimatches

In the next experiment, we try to find the optimal threshold t_1 to use in order to determine whether or not an exact match has been found among the candidates selected with the first strategy. We consider that an exact SURF match has been found between two images i and j when: $Bim(i,j) > t_1$.

For the three datasets considered, we try different values of this threshold $t_1 = \{3, 4, 5, 6\}$

	$Bim(i,j) > 3$			$Bim(i,j) > 4$			$Bim(i,j) > 5$			$Bim(i,j) > 6$		
	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.
Dat1	0.387	0.749	0.411	0.395	0.779	0.437	0.397	0.794	0.442	0.400	0.794	0.449
Dat2	0.353	0.719	0.213	0.423	0.778	0.246	0.443	0.788	0.256	0.443	0.790	0.259
Dat3	0.792	0.917	0.639	0.792	0.917	0.653	0.833	0.958	0.667	0.750	0.958	0.678

We can see that the best overall performance is obtained for $t_1 = 6$ but in the case of dataset3, the best performance is obtained with $t_1 = 5$. This means that fixing $t_1 = 5$ causes the apparition of some false exact matches while fixing $t_1 = 6$ makes us ignore some good exact matches. As one of our goals was to

recognize relevant places and buildings, we choose $t_1 = 5$. We prefer to loose some overall efficacy at expense of recognizing representative places.

We now have to combine the matches found among the candidates obtained with the first strategy with those obtained with the second strategy (these are the candidates from the visual clusters). Another threshold t_2 is needed to determine when an exact match is found among the visual clusters. We make experiments similar to the previous ones. We try different values of this threshold $t_2 = \{3, 4, 5, 6\}$

Now, when an exact match is found, two cases are possible:

- the best exact match is found among the five candidates obtained with the first strategy: in this case we propose its tags as recommended tags.

- the best exact match is found among the visual clusters obtained with the MPEG-7 descriptors: in this other case we propose the most voted tags of the cluster as recommended tags. In case of parity between two images, we choose the one belonging to the cluster with a higher Bim_{vc}.

If no exact match is found, we propose tags as in section 4.5, i.e. we ignore the SURF features. Therefore, adding those new candidates should not cause great changes in the overall performance for DAT1 and DAT2 because few exact matches are found. On the other hand, the performance for DAT3 should increase significatively. More precisely, the results of this section should show an increase in the performance of the classification. In fact, even if an exact match is found in one of the clusters, the tags proposed by that cluster are the same, the difference is that the those tags are heavily weighted and are always proposed as recommended tags.

	$Bim(i,j) > 3$			$Bim(i,j) > 4$			$Bim(i,j) > 5$			$Bim(i,j) > 6$		
	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.	Class	Said C.	T.A.
Dat1	0.406	0.758	0.438	0.415	0.776	0.446	0.405	0.791	0.449	0.399	0.788	0.447
Dat2	0.403	0.774	0.250	0.429	0.782	0.253	0.439	0.784	0.255	0.441	0.788	0.256
Dat3	0.960	1.0	0.728	0.960	1.0	0.739	0.917	0.960	0.700	0.833	0.960	0.680

We can see that the results remain quite identical for DAT1 and DAT2 while we obtain impressive results for dataset3 in the cases of $t_2 = 3$ and $t_2 = 4$. This methodology has been succesful in 24 of the 25 images present in Dataset3. In fact, the bad classified image corresponds to a case in which no an exact match has been found and then SURF features have been ignored. Once again, we aware that fixing $t_2 = 4$ causes some false exact matches.

We consider the efficacy achieved for DAT3 satisfactory and we consider our initial goal of recognizing popular places achieved.

We now show the strategy followed in order to recommend tags:

find the 5 best candidates with Strategy 1
for each image $B \in candidates1$ **do**
  compute $bimatches(A, B)$
  if $bimatches(A, B) > 5$ **then**
    for $t \in tags(B)$ **do**
      $score(t) = 100$
    end for
  end if
end for

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters C_j with $p(C_1) > 0$
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    if $p(t_i|C_j) > 0.1$ **then**
      $score(t_i) = score(t_i) + w_d * \sum_{j=1}^{3}$ occurrences of t_i in C_j
    end if
  end for
  for each $vc \in visualclusters$ **do**
    find the 3 best candidates with Strategy 2
    for each image $B \in candidates2$ **do**
      compute $bimatches(A, B)$
      if $bimatches(A, B) > 5$ **then**
        for $t \in tags(vc)$ **do**
          $score(t) = score(t) + 100 *$ occurrences of t in vc
        end for
      end if
    end for
  end for
end for
recommend tags with the highest score

Chapter 6

Improvements

6.1 Tag cooccurrence

We have realized that some tags tend to appear together. In the article 'Flickr Tag Recommendation based on Collective Knowledge', Brkur Sigurbjrnsson and Roelof van Zwol propose a recommending strategy based on the fact that tags that usually appear together are related. Hence, we can use the tag cooccurrence as a measure of semantic similarity. For example, tags like 'forest' and 'tree' often appear together, i.e. they have a great cooccurrence. We think we can use a technique based on tag cooccurrence in order to improve the global efficacy of our tag recommending sytem. The idea of our technique is that if we are to propose a set of tags T, the tag t which has the highest cooccurrence with the tags composing T is likely to be a good recommendation.

We define:

Let $Cooc(t_i, t_j)$ be the tag cooccurrences between the tags t_i and t_j
Let $T = \{t_1, .., t_n\}$ be the set of tags found with MPEG-7 or SURF features for a given image,
Let $score(t_j, T) = \sum_{i=1}^{n} Cooc(t_i, t_j)$ with $t_i \in T$

Finding the Most Coocurrent Tags

We use the following method in order to find tags related with the set T:

for $j \in \{1..m\}$ **do**
find the tag $t_j \in vocabulary$ such that $score(t_j, T) = \max \sum_{i=1}^{n} Cooc(t_i, t_j)$
end for
include the tags with the highest score in the set T

One of the main reasons that forces us to adopt this technique is that many times, we don't propose enough tags. Nevertheless, one of our goals was to use mainly visual features in order to recommend tags. This technique is obviously not related with visual features. Therefore we use it as a way to complete the tags obtained with visual features, but not to replace them.

In fact, as we explained before our goal was to propose 10 tags for each image. There are two cases in wich not enough tags are proposed:

- when an exact match is found among the candidates found with the first strategy and this image has less than 10 tags.
- when the visual clusters found with MPEG-7 descriptors have low $p(C, t)$ for the tags they propose

In both of these cases we complete the tags recommended up to ten with this method.

We now show some examples of tags found with this method:

Example 1:
tags found with an exact SURF match: $\{barcelona, sagradafamilia, gaudi, spain\}$
tags added with this method: $\{espaa, architecture, catalunya, spanien, bcn, catalonia\}$
final 10 tags: $\{barcelona, sagradafamilia, gaudi, spain, espaa, architecture, catalunya, spanien, bcn, catalonia\}$

Example 2:
9 tags found with visual clusters: $\{bluesky, blue, sky, beach, sand, woods, hiking, rockyrunnerbird, nature\}$
6 tags found with this method: $\{clouds, sunset, green, trees, landscape, flowers\}$
final 10 tags: $\{bluesky, blue, sky, beach, sand, woods, hiking, rockyrunnerbird, nature, clouds\}$

As we said before, our goal was to use a content-based approach to train our recommender. We use this method in order to complete the set of tags proposed up to ten, therefore we are not intersted in comparing the improvement of the efficacy of the recommender.

6.2 Implementation Improvement

In order to perform the similarity searchs in a more efficient way, we use an index structure called M-tree. This structure allows us to resolve similarity queries in metric spaces. In this section, we give a definition of metric spaces and we explain briefly how we use the M-trees in our application.

6.2.1 Metric Spaces

A metric space is an ordered pair (M, d) where M is a set and d is a metric on M, i.e a distance function

$$d : M \times M \to \mathbb{R} \tag{6.1}$$

such that for any x, y and z in M

$$d(x, y)0 \quad \text{(non-negativity)} \tag{6.2}$$
$$d(x, y) = 0 \text{ if and only if } x = y \quad \text{(identity of indiscernibles)} \tag{6.3}$$
$$d(x, y) = d(y, x) \quad \text{(symmetry)} \tag{6.4}$$
$$d(x, z)d(x, y) + d(y, z) \quad \text{(triangle inequality)} \tag{6.5}$$

Here is a more compact definition of metric spaces:

$$d(x, y) = 0 \text{ if and only if } x = y \tag{6.6}$$
$$d(x, z)d(x, y) + d(z, y) \tag{6.7}$$

In our case, we can consider that we have a metric space for each MPEG-7 descriptor that we use. Therefore we have six different metric spaces. In each of the six cases, the set M is formed by the MPEG-7 features extracted from the training sets and the distance functions are the ones explained in chapter 2.

We also consider another metric space, the one formed by the SURF keypoints. In order to work with this metric space, we consider the distance function between two keypoints to be the euclidean distance between their descriptor vector.

6.2.2 Use of the M-tree structure

We have identified seven different metric spaces in our dataset, therefore using this structure is suitable for us. In fact, M-trees are usually used for content-based retrieval but they have also been successfully used in other research fields like Data Mining and Case-Based Reasoning.

The version used of the M-tree allows nearest neighbors queries, which is perfect for us. In fact the query consists of an object (an MPEG-7 feature or a SURF keypoint) and the amount of nearest neighbors we want to retrieve. When we start the application, we must insert all the features extracted from the training sets in their respective M-tree. The insert operation takes long but it must be done only once. When we use this structure, the computational time of the MPEG-7 features similarity searchs passes from more than ten seconds to less than one second for each analyzed image. On the other hand, the similarity search of the SURF keypoints is dramatically speeded-up. as we said in section 5.1, the linear search of the nearest keypoint in Dataset1 composed of 1333 images takes a few minutes while with the use of an M-tree it takes less than a second. Nevertheless the feature extraction time remains the same and is now the real bottleneck of our application.

Chapter 7

Big Scale Application

At this point of the project, we consider we have reached a satisfactory performance in the tags we are recommending. We think that building a bigger scale application must increase the accuracy of proposed tags. The idea is that more types of landscapes and more buildings are present in a bigger training set.

7.1 Cleaning of the dataset

We still work with the CoPhir dataset, we think working with 100000 images is appropiate. As we did with the datasets used previously, we proceed to filter both images and tags that form the training set as follows:

- We choose randomly 15000000 images, we have 695115 different tags

- We get the tags of the 15000000 images

- we now have a vocabulary formed by 695115 different tags

- We keep the images having: 3 <= tags <= 12

- We now have 5967888 images

- We keep the images having: $comments > 6$, $views > 110$ and $faves > 0$

- We now have a set of 100127 images

- We keep the tags having: $100 < appearances < 60000$

- We now have a vocabulary of 31785 tags

7.2 Textual Clustering

We now need to proceed with the textual clustering as we did for the previous datasets. If we want to use the software k-metis, we need to represent the dataset as a graph where the nodes are the images and the weights of the arcs are the semantic similarities between them. This means that we have to compare each image with all the others and, for such a big dataset, the computational cost is too high. Therefore we have to find another way to proceed with the textual clustering. We decide to use a k-means clustering algorithm in order to obtain textual clusters. The distance used remains the semantic similarity. We want to obtain

textual clusters of approximately 50 images, hence We fix $k = 2000$. At the end of this process we have 2000 different textual clusters.

7.3 Visual Clustering

We now use another k-means clustering algorithm in order to obtain visual clusters. This time we want to obtain visual clusters formed by approx. 20 images. As the training set is bigger than in the previous datasets, we consider that visual clusters must be bigger. For each of the MPEG-7 descriptors, we obtain 4800 clusters containing approx. 20 images each. We now follow the exact same procedure explained in section 4.5.5 in order to filter these visual clusters. For each visual cluster C considered, we also obtain the value $p(t|C)$ for each of the tags t that the visual cluster recommends.

7.4 SURF keypoints

We extract the SURF keypoints for every image in the dataset. But we realize that even if the datsaet considered is huge there are not so many images of representative buildings. For example:

- only 25 images contain the tag 'empirestatebuilding'
- only 14 images contain the tag 'colosseo'
- only 12 images contain the tag 'goldengate'
- 0 images contain the tag 'fontanaditrevi'
- 0 images contain the tag 'puertadealcala'

Therefore, if we want our recommender to identify representative places or buildings, we must introduce new images of these places in the dataset. We use FlickrJ with this pourpose. FlickrJ is a java api which allows us to perform searchs in the Flickr. In order to obtain images from Flickr, we must send a textual query such as { 'colosseo','rome'} and specify the amount of images we want to retrieve. We observe that the images retrieved are usually representative of the buildings which make them suitable for our pourpose.

For each of the buildings or places we want to recognize, we follow the next procedure:

- we obtain 100 images with the Flickrj api
- we obtain their tags and order them by number of appearances
- we extract the SURF keypoints of every retrieved image
- we filter the extracted keypoints following the same idea explained in section 5.2:

We follow the next procedure in order to filter these keypoints:

for each image $A \in$ 100 retrieved images from Flickr **do**
  for $k \in keys_A$ **do**
    compare to all the other keypoints extracted from the 100 retrieved images
    if a match is produced with an image B with distance $< T$ **then**
      keep k
    else
      delete k
    end if
  end for
end for

This way, we are sure to have significative SURF keypoints of the buildings or places we want to recognize.

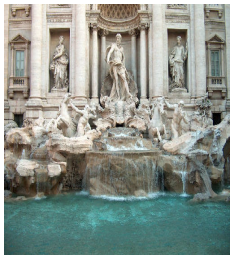
We decide to include the following representative buildings or places:

ROME

- Campidoglio - Fontana di Trevi - Colosseo - San Pietro

ROME

Campidoglio



Fontana di Trevi

Colosseo

San Pietro

Figure 7.1: Examples of Recognized Images from Rome

MADRID

- Plaza Mayor - Puerta de Alcal - Puerta del Sol

MADRID

Plaza Mayor

Puerta de Alcala

Puerta del Sol

Figure 7.2: Examples of Recognized Images from Madrid

MARRAKECH

- Bab Agnaou Gate - Koutoubia Mosque

MARRAKECH

Bab Agnaou Gate  Koutoubia Mosque 

Figure 7.3: Examples of Recognized Images from Marrakech

ISTANBUL

- Blue Mosque - Galata Tower

ISTANBUL

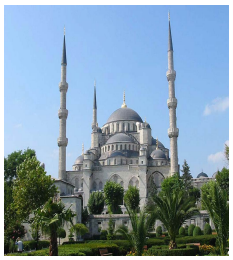
Blue Mosque  Galata Tower 

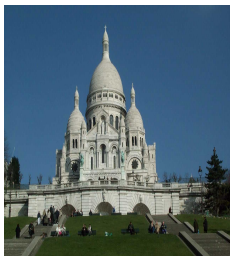
Figure 7.4: Examples of Recognized Images from Istanbul

PARIS

- Notre Dame - Sacre Coeur - Eiffel Tower

PARIS

Notre Dame 

Sacre Coeur 

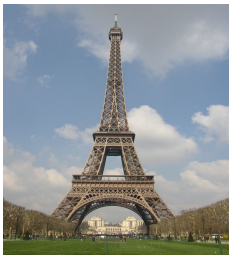
Eiffel Tower 

Figure 7.5: Examples of Recognized Images from Paris

PISA

- Leaning Tower of Pisa

PISA

Leaning Tower 

Figure 7.6: Examples of Recognized Images from Pisa

7.5 Strategy followed

In order to recommend tags, we follow a very similar strategy to the one explained in section 5.5, we show here the new procedure:

find the 10 best candidates with Strategy 1
for each image $B \in candidates1$ **do**
  compute $bimatches(A, B)$
  if $bimatches(A, B) > 5$ **then**
    for $t \in tags(B)$ **do**
      $score(t) = 100$
    end for
  end if
end for

for $d \in descriptors$ **do**
  find the 3 nearest visual clusters C_j with $p(C_1) > 0$
  for $t_i \in \bigcup_{j=1}^{3} tags(C_j, d)$ **do**
    if $p(t_i|C_j) > 0.1$ **then**
      $score(t_i) = score(t_i) + w_d * \sum_{j=1}^{3}$ occurrences of t_i in C_j
    end if
  end for
  for each $vc \in visualclusters$ **do**
    find the 5 best candidates with Strategy 2
    for each image $B \in candidates2$ **do**
      compute $bimatches(A, B)$
      if $bimatches(A, B) > 5$ **then**
        for $t \in tags(vc)$ **do**
          $score(t) = score(t) + 100 *$ occurrences of t in vc
        end for
      end if
    end for
  end for
end for
recommend tags with the highest score

7.6 Examples

We show here some examples of tags recommended with this technique:

REC. TAGS:
water
sky
clouds
blue
landscape
sea
green
reflection
river
flower

REC. TAGS:
fontanaditrevi
roma
rome
italy
trevifountain
trevi
europe
people
fuentes
barroco

REC. TAGS:
architecture
night
sky
city
landscape
clouds
sunset
blue
tree
sun

REC. TAGS:
red
flower
yellow
orange
sunset
naturefinest
light
insect

REC. TAGS:
sunset
sky
red
flower
clouds
yellow
orange
night
sun
sunrise

REC. TAGS:
marrakech
morocco
gate
babagnaou
maroc
bab
marrakesh
agnaou
marokko
travel

Chapter 8

Conclusions

We have implemented a tag recommending system in order to improve the quality of tags in web pages such as Flickr or ImageShack. Our recommender is suitable for these pages because the user will not need to write tags anymore, making the user's experience less frustrating. Our recommender can also be used in a hidden way in order to enrich the text associated to some multimedia content. Therefore, better retrieval systems can be implemented in these pages.

In this work, we have followed a content-based approach in order to recommend tags. The content-based approach is a large field of research because this approach depends on the kind of data we work with. In our case, working with image features has been challenging because many image processing algorithms take too long and can't be used in order to propose tags in an acceptable time. Our work has many common points with the techniques used in content-based image retrieval. In fact, we have implemented a search engine based on Mpeg-7 features. A combination of Mpeg-7 descriptors of the query image is used to retrieve similar images. In other works, image features such as Mpeg-7 features are used in order to classify images in a reduced set of categories while we established relations between these features and tags. In fact, extracting semantic meaning from images is an open area in research and has been very challenging.

We have also worked with SURF features. These features are widely used in object recognition and allow us to identify representative places such as famous buildings or squares. We tried to obtain an optimal combination of both Mpeg-7 and SURF features in order to recommend tags accurately.

We have built a large scale application in order to train our recommendation engine. We have proposed and used innovative scalable methods in order to train our recomender system and to compare the new images with the training set. It has been especially challenging in the case of SURF features because the matching process has a high computational cost. For the achievement of our experiments, we have worked with the Cophir dataset, which is a large collection of images (over 100 million) taken from Flickr. This dataset contains information about the image posted by users (tags, views, comments...) which makes the dataset suitable for content-based image retrieval research.

But this dataset is very noisy and many images are bad tagged. Also the vocabulary of tags is huge and the tags are very sparse. Therefore, we filtered both the images and the tags in order to avoid an excessive noise in the dataset. In fact, the noise in the dataset, specially the sparsity of tags has been one of the biggest difficulties we have found while trying to achieve our pourpose.

In this thesis, we propose an innovative scalable tag recommender based on visual features of the images. We have used Mpeg-7 and SURF descriptors which are global and local features respectively. We present a combination of these features in order to recommend tags. We have adopted a visual clustering technique in order to solve the scalability problem. The images in the training set were firstly divided in textual clusters with the software k-metis using a technique based on the similarity between the tags of the different images. A second segmentation was performed with a k-means algorithm based on the Mpeg-7 descriptors. The

combination of both textual and visual clustering leads to obtain clusters of images having a strong semantic relatedness. With the aim of recommending tags for a given image, we follow an innovative strategy that combines the use of SURF features and the visual clusters obtained with the textual+visual clustering.

We propose a cleaning technique that allows to identify the cases in which there is a strong relation between visual clusters and tags. With this cleaning, both the efficacy and the efficiency of our recommender was increased. We also propose a candidate selection strategy that allows us to perform the SURF matching process with the best candidates among the whole dataset. We use of the M-tree structure that allows us to perform similarity searchs in our filtered dataset of both MPEG-7 and SURF features, the use of this structure increases dramatically the efficiency and allows us to recommend tags in an acceptable time.

We also propose some metrics in order to evaluate the accuracy of our recommender and to compare it with other techniques.

Our recommender could be useful for other applications. In fact, the SURF matching process proposed is especially interesting because a software based on this process could be integrated in mobile phones equipped with a digital photo camera. For example tourist guides could be implemented where information is displayed when taking a photo of a representative building.

References

[1] Yong Man Ro, Munchurl Kim, Ho Kyung Kang, B.S. Manjunath, and Jinwoong Kim *MPEG-7 Homogeneous Texture Descriptor.*

[2] Chee Sun Won, Dong Kwon Park and Soo-Jun Park *Efficient Use of MPEG-7 Edge Histogram Descriptor.*

[3] Wong Ka Man *Content Based Image Retrieval Using* MPEG-7 *Dominant Color Descriptor.*

[4] Laila H.Shoukry *Visualizing MPEG-7 Color Layout and Edge Histogram Descriptors for CBIR Systems.*

[5] David G. Lowe *Object Recognition from Local Scale-Invariant Features.*

[6] David G. Lowe *Distinctive Image Features from Scale-Invariant Keypoints.*

[7] Ballard Blair and Chris Murphy *Difference of Gaussian Scale-Space Pyramids for* SIFT *Feature Detection.*

[8] Dan Huttenlocher *Computer Vision slides.*

[9] Herbert Bay, Andreas Ess, Tinne Tuytelaars and Luc Van Gool *Speeded-Up Robust Features (*SURF*).*

[10] Michal Batko, Fabrizio Falchi, Claudio Lucchese, David Novak, Raffaele Perego, Fausto Rabitti, Jan Sedmidubsky, Pavel Zezula *Building a Web-scale Image Similarity Search System.*

[11] Lei Wu, Mingjing Li, Zhiwei Li, Wei-Ying Ma and Nenghai Yu *Visual Language Modeling for Image Classification.*

[12] Aiden R. Doherty, Ciarn Conaire, Michael Blighe, Alan F. Smeaton and Noel E. O'Connor *Combining Image Descriptors to Effectively Retrieve Events from Visual Lifelogs.*

[13] Brkur Sigurbjrnsson, Roelof van Zwol *Flickr Tag Recommendation based on Collective Knowledge.*

[14] Tiziano Fagni, Fabrizio Falchi and Fabrizio Sebastiani *Adaptive Committees of Feature-specific Classifiers for Image Classification.*

[15] Boris Ruf, Effrosyni Kokiopoulou, and Marcin Detyniecki *Mobile museum guide based on fast* SIFT *recognition.*

[16] George Karypis and Vipin Kumar *METIS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices.*

[17] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic and Andrew Zisserman *Object retrieval with large vocabularies and fast spatial matching.*

[18] J. Jeon, V. Lavrenko and R. Manmatha *Automatic Image Annotation and Retrieval using CrossMedia Relevance Models.*

[19] Lyndon Kennedy, Mor Naaman, Shane Ahern, Rahul Nair and Tye Rattenbury *How Flickr Helps us Make Sense of the World: Context and Content in Community-Contributed Media Collections.*

[20] Pavel Serdyukov, Vanessa Murdock and Roelof van Zwol *Placing Flickr Photos on a Map.*

[21] David Crandall, Lars Backstrom, Daniel Huttenlocher and Jon Kleinberg *Mapping the World's Photos.*